

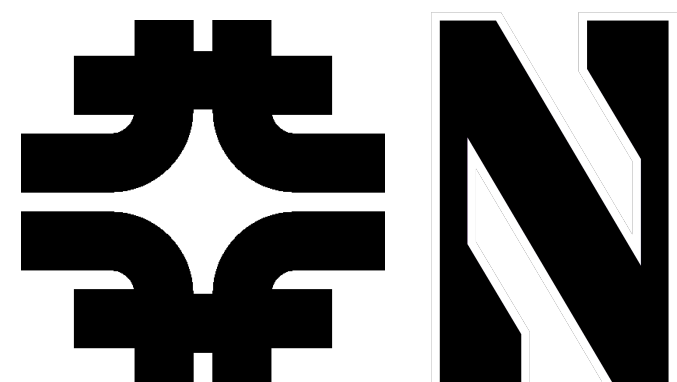
(Efficient) ML in trigger (real-time system) deployment

Nhan Tran

Fermilab/Northwestern ECE

A14EIC workshop

September 9, 2021

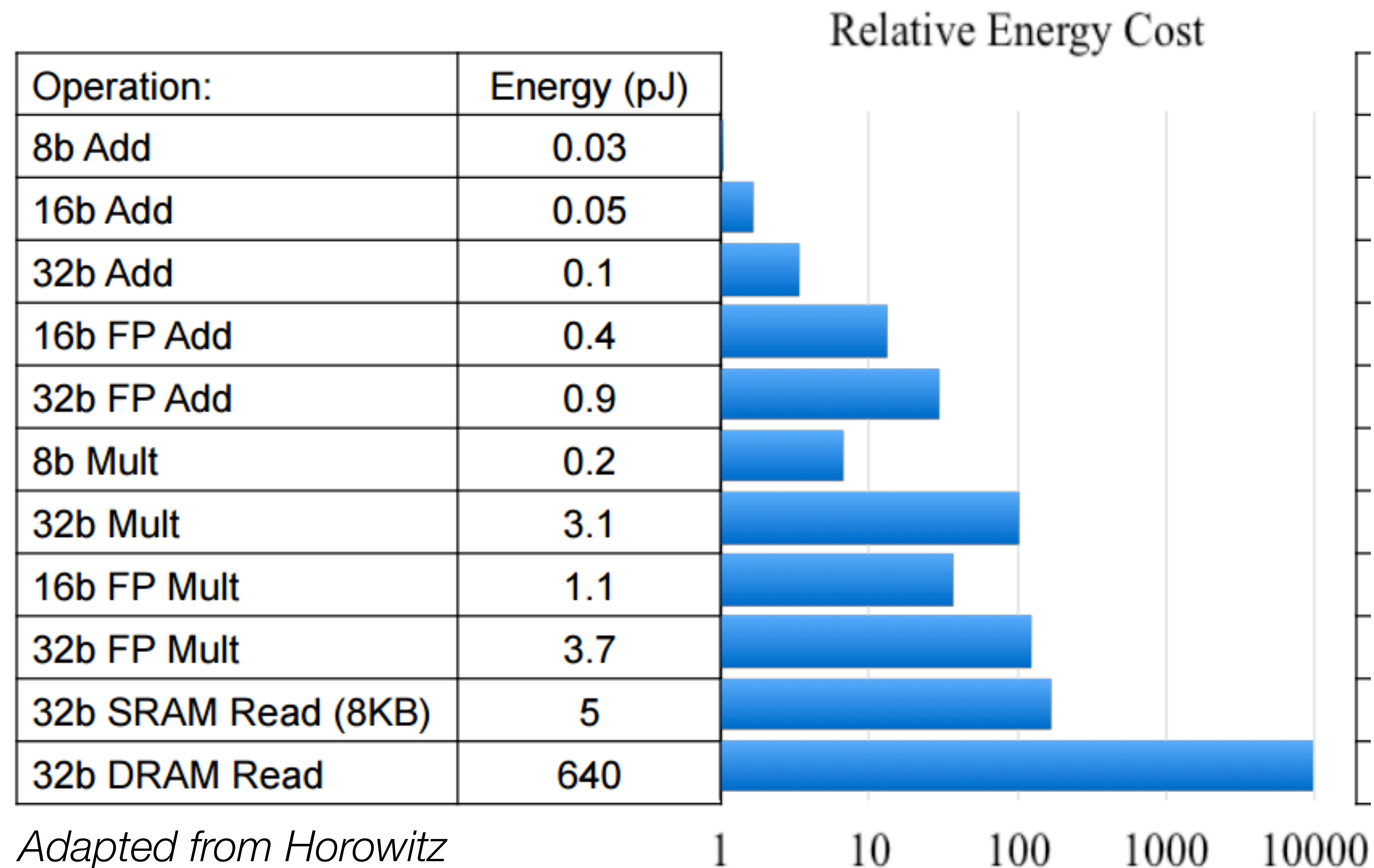


Introduction

- ML is powerful paradigm for **optimal and automated** control and calibration
 - real-time control for detectors = online data processing/selection (triggering)
 - real-time control for accelerators = efficient beam operation
- ML is computationally expensive — but less expensive than data movement
- Plan for this talk
 - *Will not make the case for ML or real-time processing, hopefully you've already been convinced!
 - **Discuss system architecture considerations for ML deployment in real-time systems**
 - **Present a few examples - across the full spectrum**
 - **Open-ended thoughts on future challenges**

Outline

- **System level goals and constraints**
 - Bandwidth, latency, and processing technologies
- **ML in trigger — techniques and examples**
 - Optimized algorithms, implementations, tools (see [Dylan's talk](#))
 - Examples:
 - Reconfigurable ASICs (see [Farah's talk](#))
 - CMS Muon Trigger
 - SONIC
- **Open challenges and food for thought**
 - Continuous learning (w/accelerator control example)
 - Training samples – online learning, transfer learning, domain adaptation

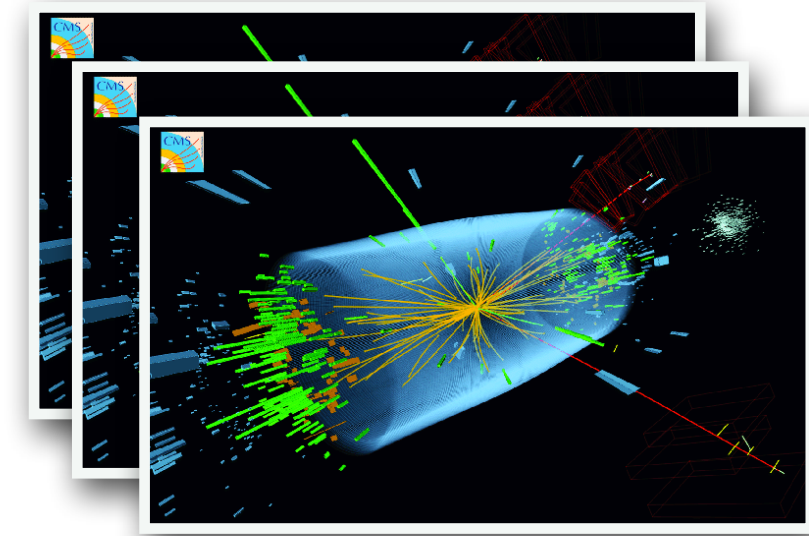


Moving data is expensive

Universal struggle — reduce data as close to the source as possible **vs** important data FOMO

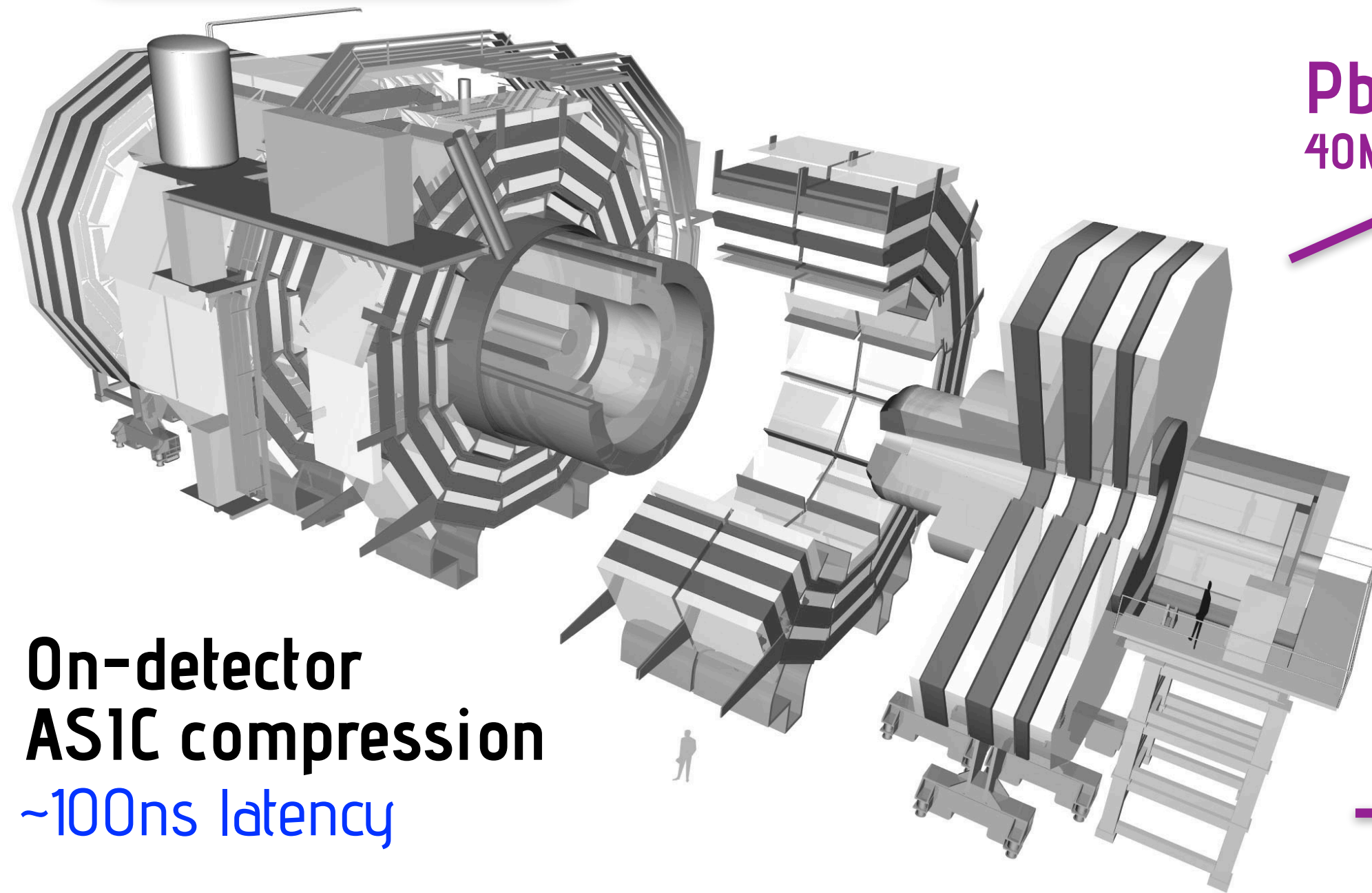
ML gets us closer to enjoying the best of both worlds?

An example of an extremely heterogeneous real-time system



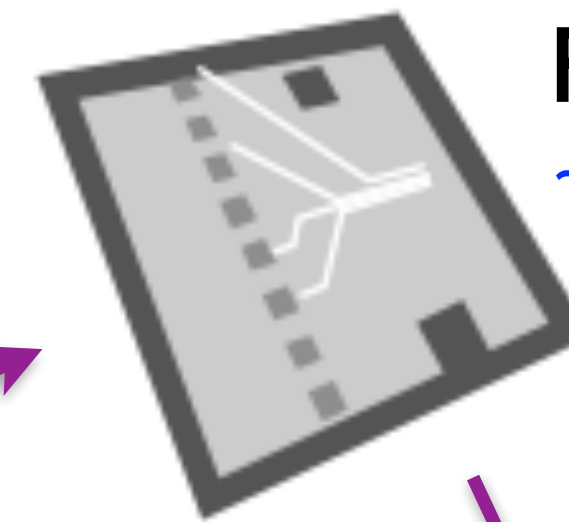
CMS Experiment

40MHz collision rate
~1B detector channels



On-detector
ASIC compression
~100ns latency

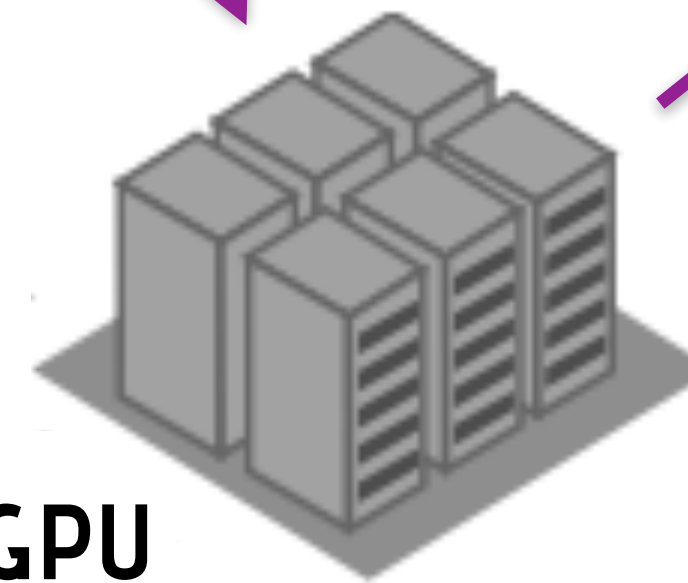
Pb/s
40MHz



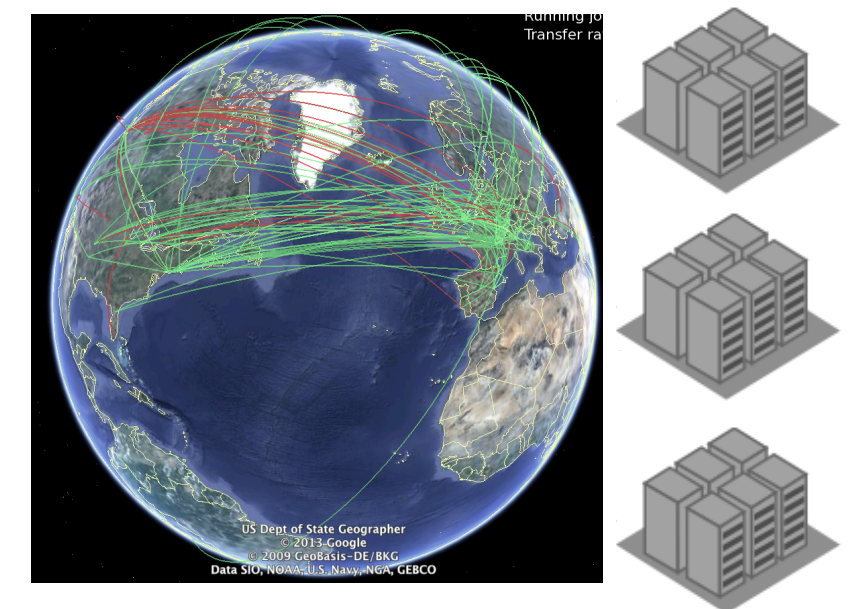
FPGA filter stack
~ μ s latency

10s Tb/s
100s kHz

On-prem CPU/GPU
filter farm
~100 ms latency

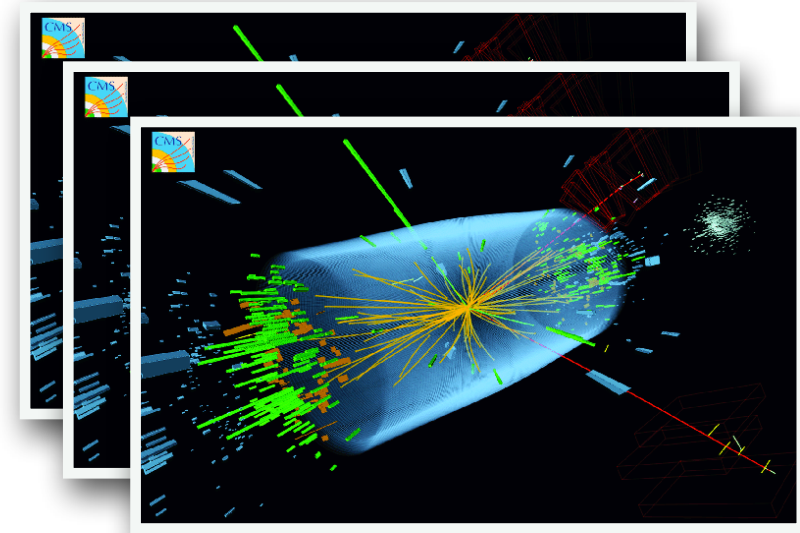


10s Gb/s
~5 kHz



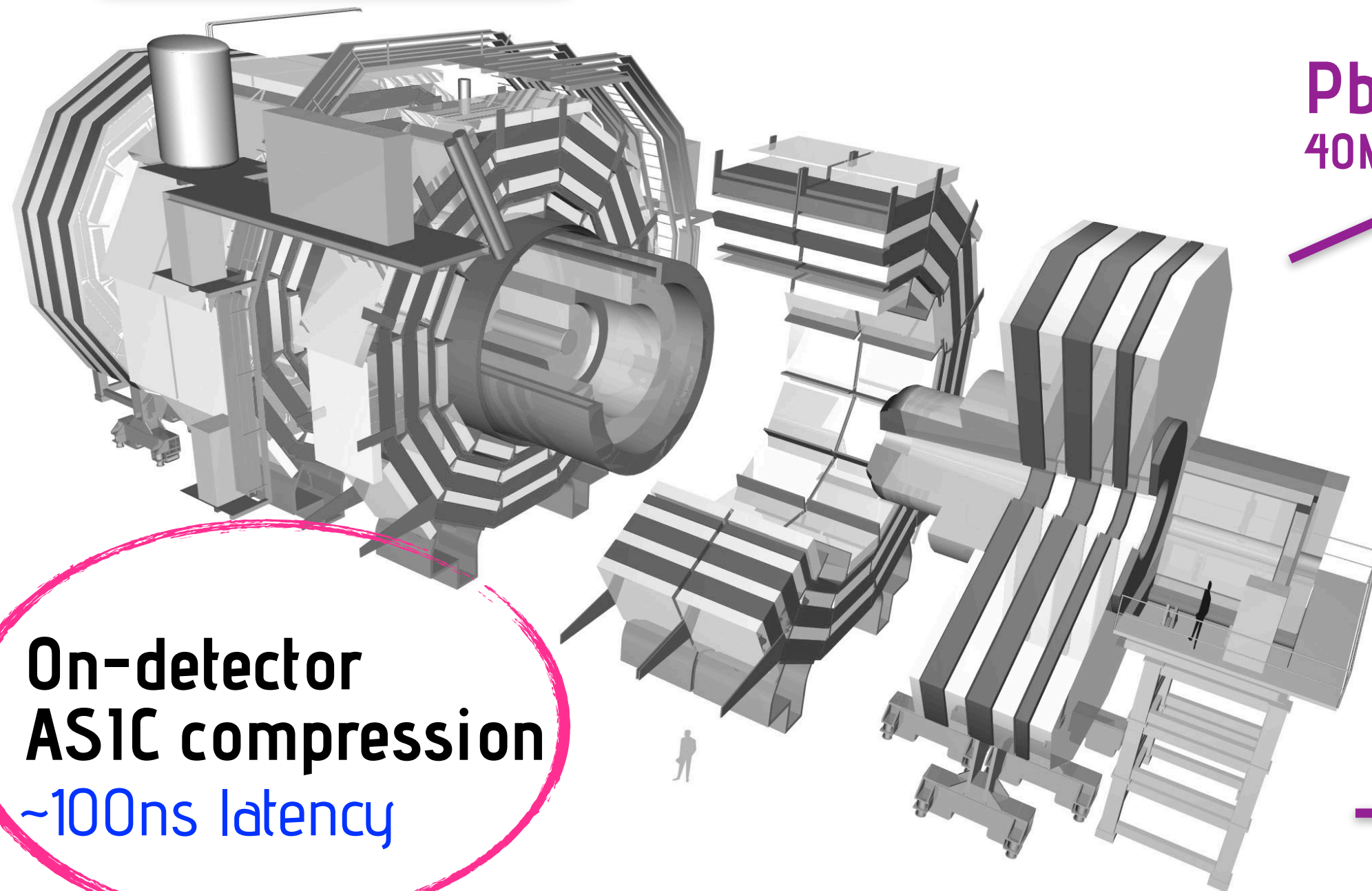
Worldwide
computing grid
Exabyte-scale
datasets

An example of an extremely heterogeneous real-time system



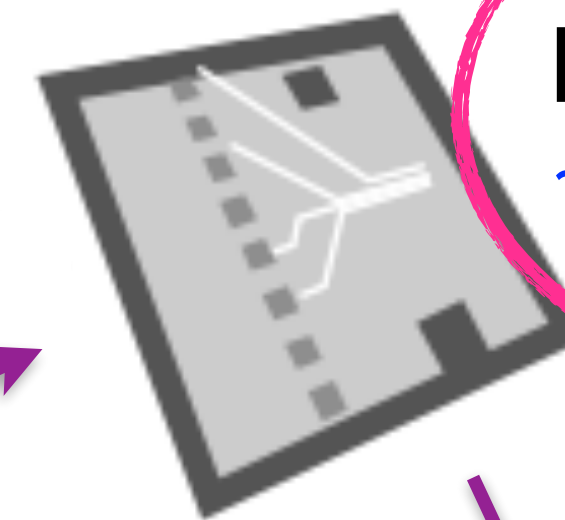
CMS Experiment

40MHz collision rate
~1B detector channels



On-detector
ASIC compression
~100ns latency

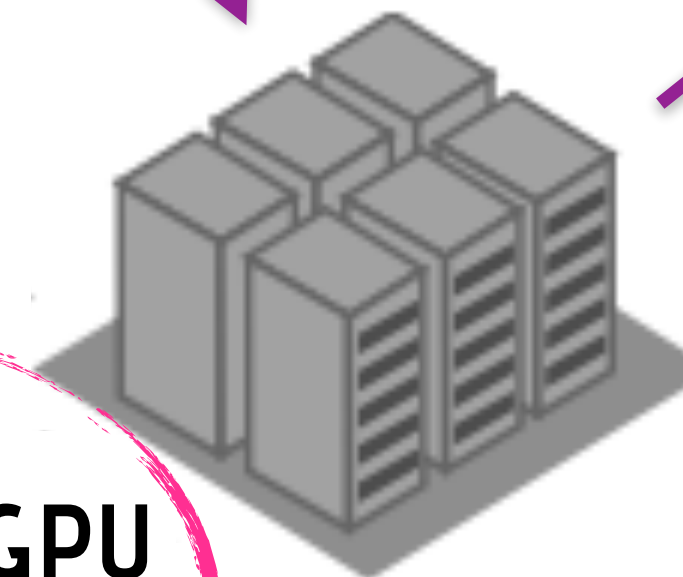
Pb/s
40MHz



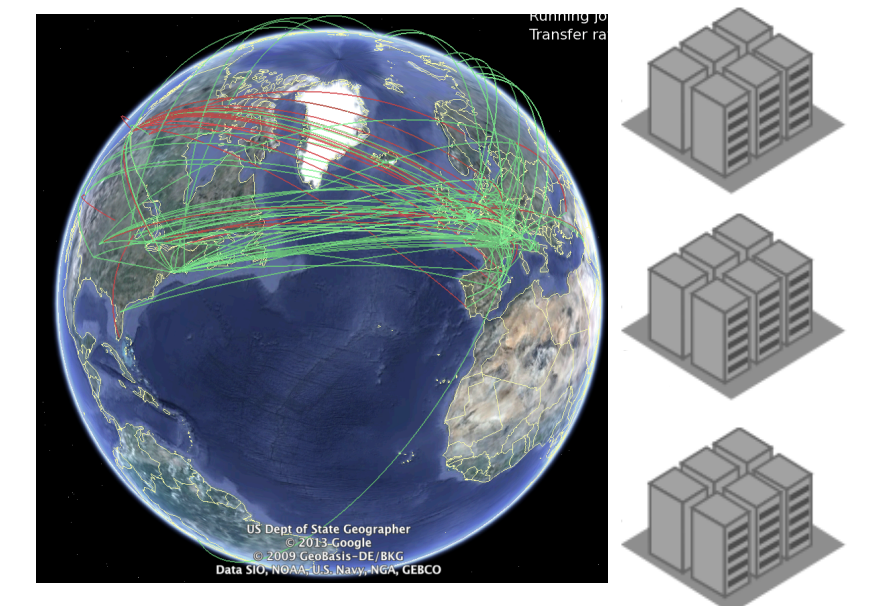
FPGA filter stack
~ μ s latency

10s Tb/s
100s kHz

On-prem CPU/GPU
filter farm
~100 ms latency



10s Gb/s
~5 kHz

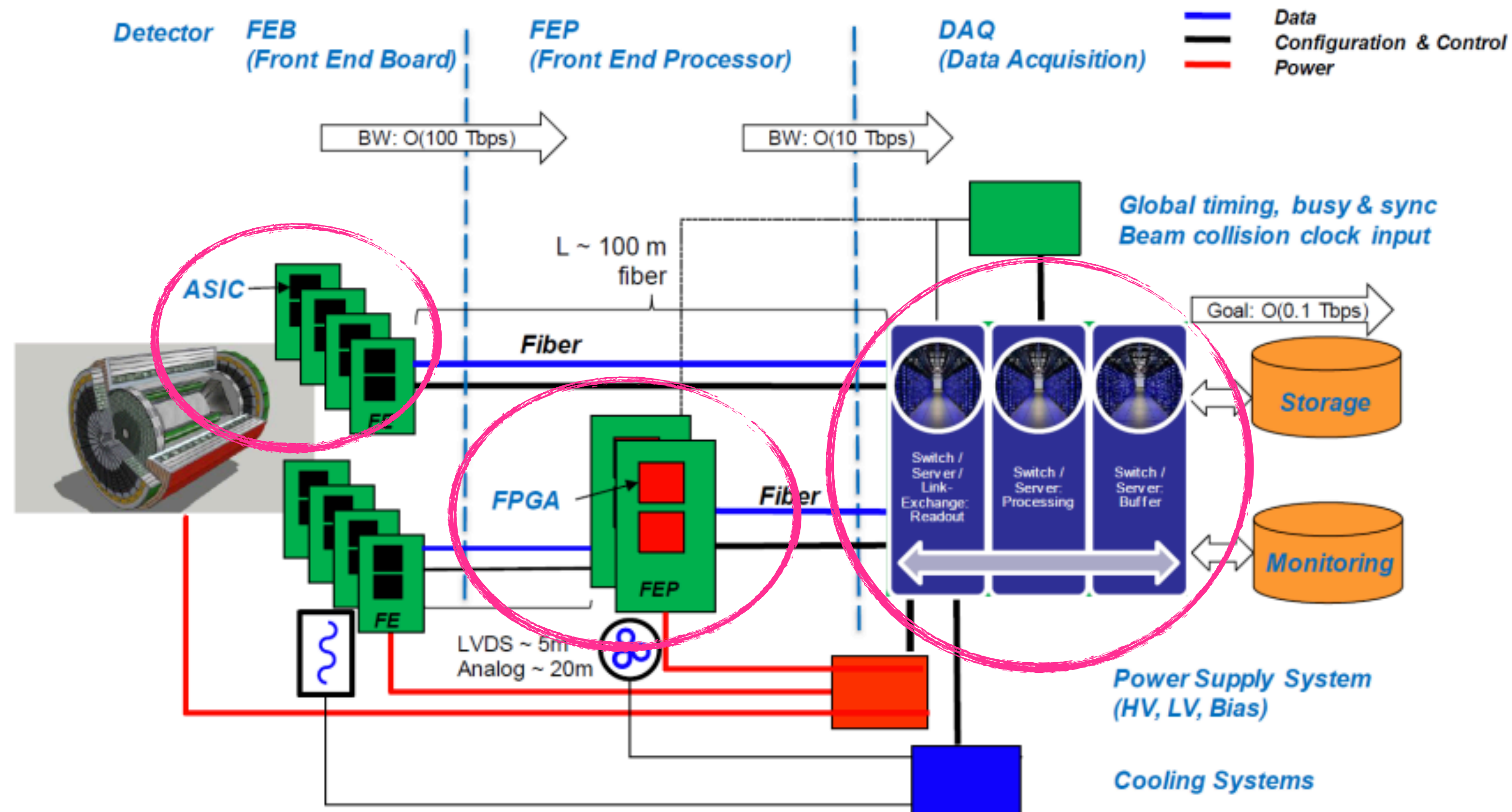


Worldwide
computing grid
Exabyte-scale
datasets

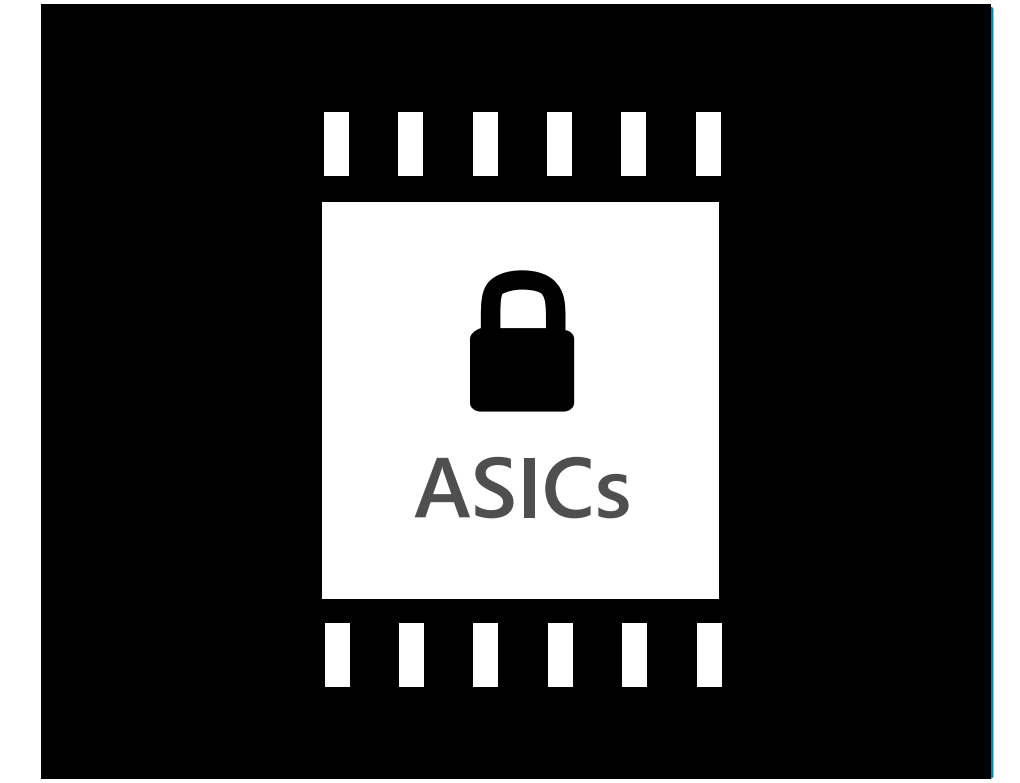
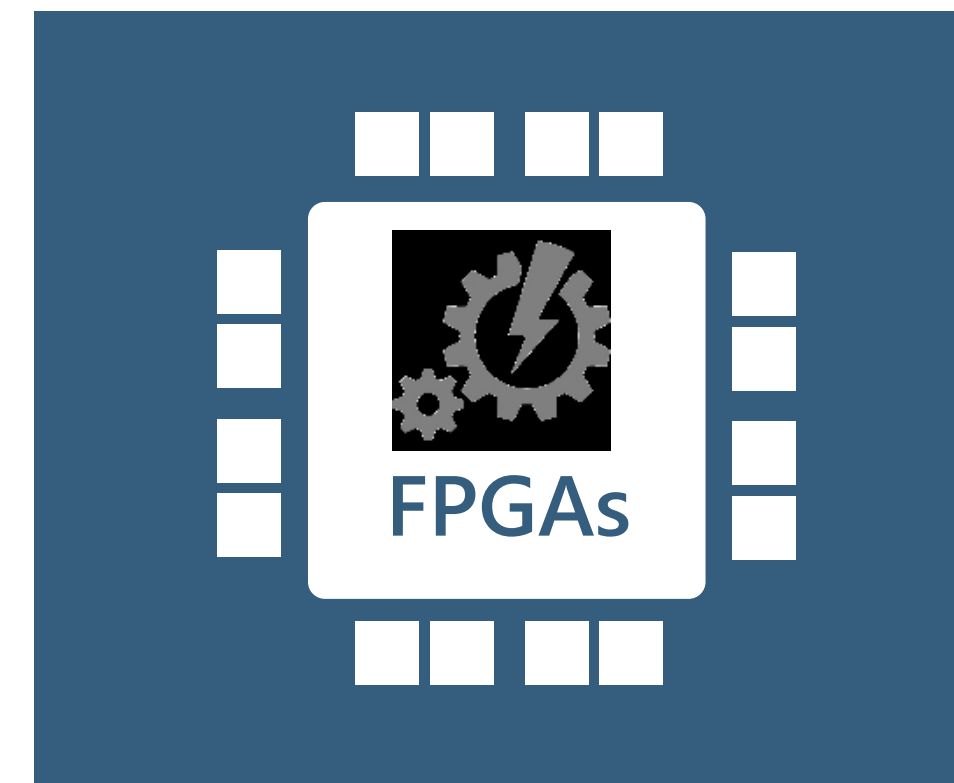
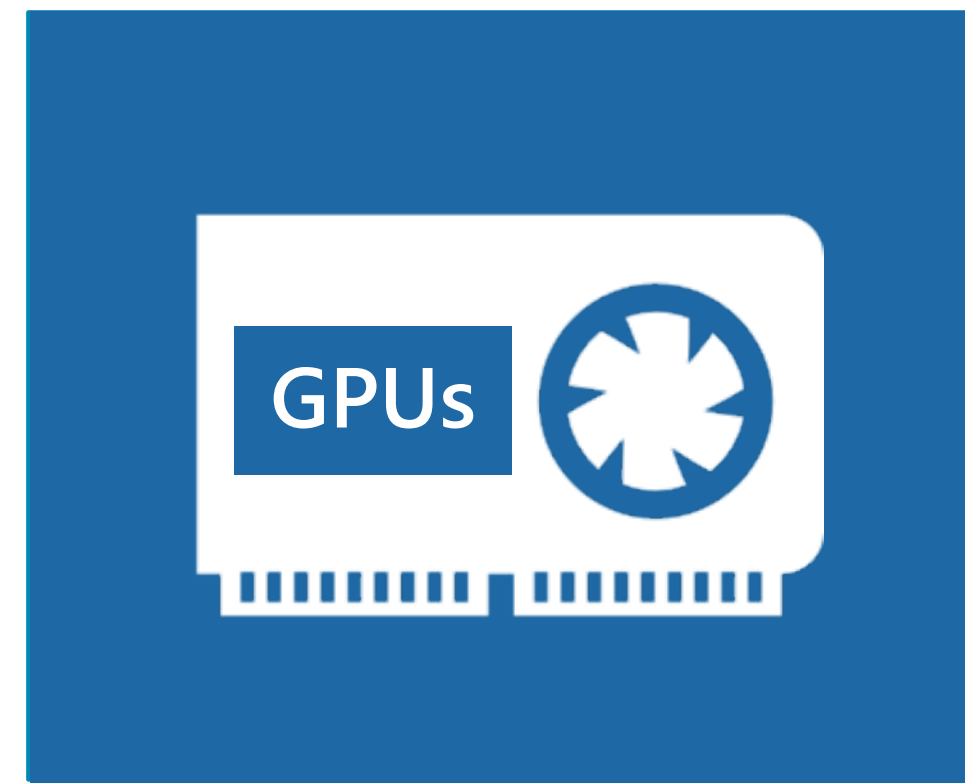
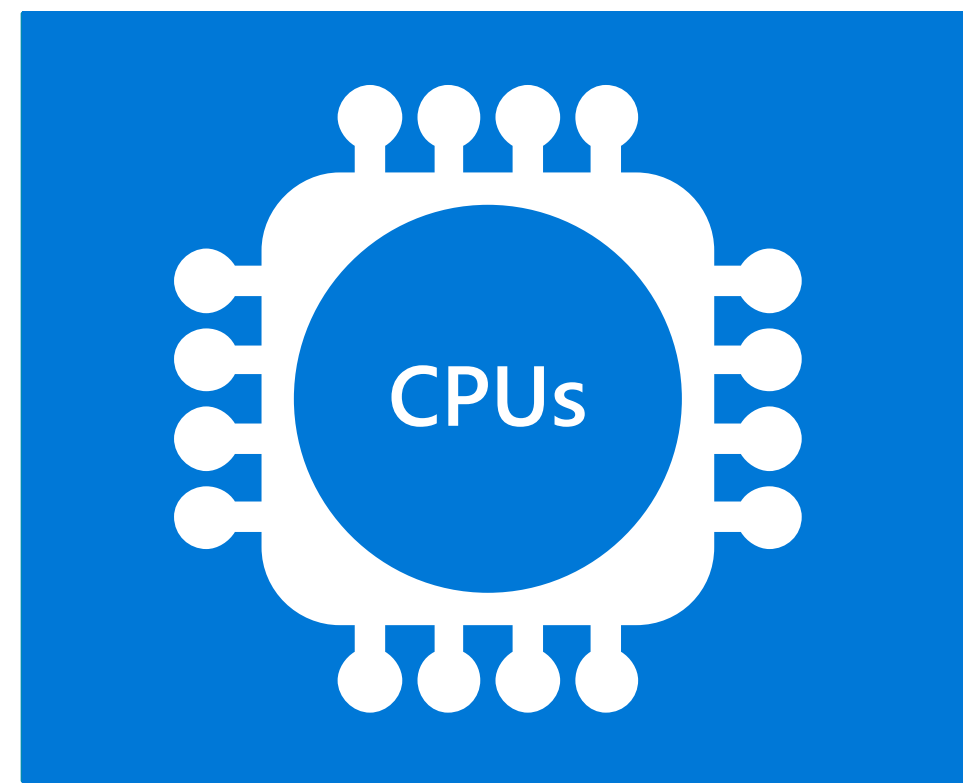
ASIC, FPGA

FPGA, TPU, GPU, CPU

Digitizer BW: 10^2 Tbps \rightarrow Readout BW: 10^2 Tbps \rightarrow Storage: 10^{-1} Tbps

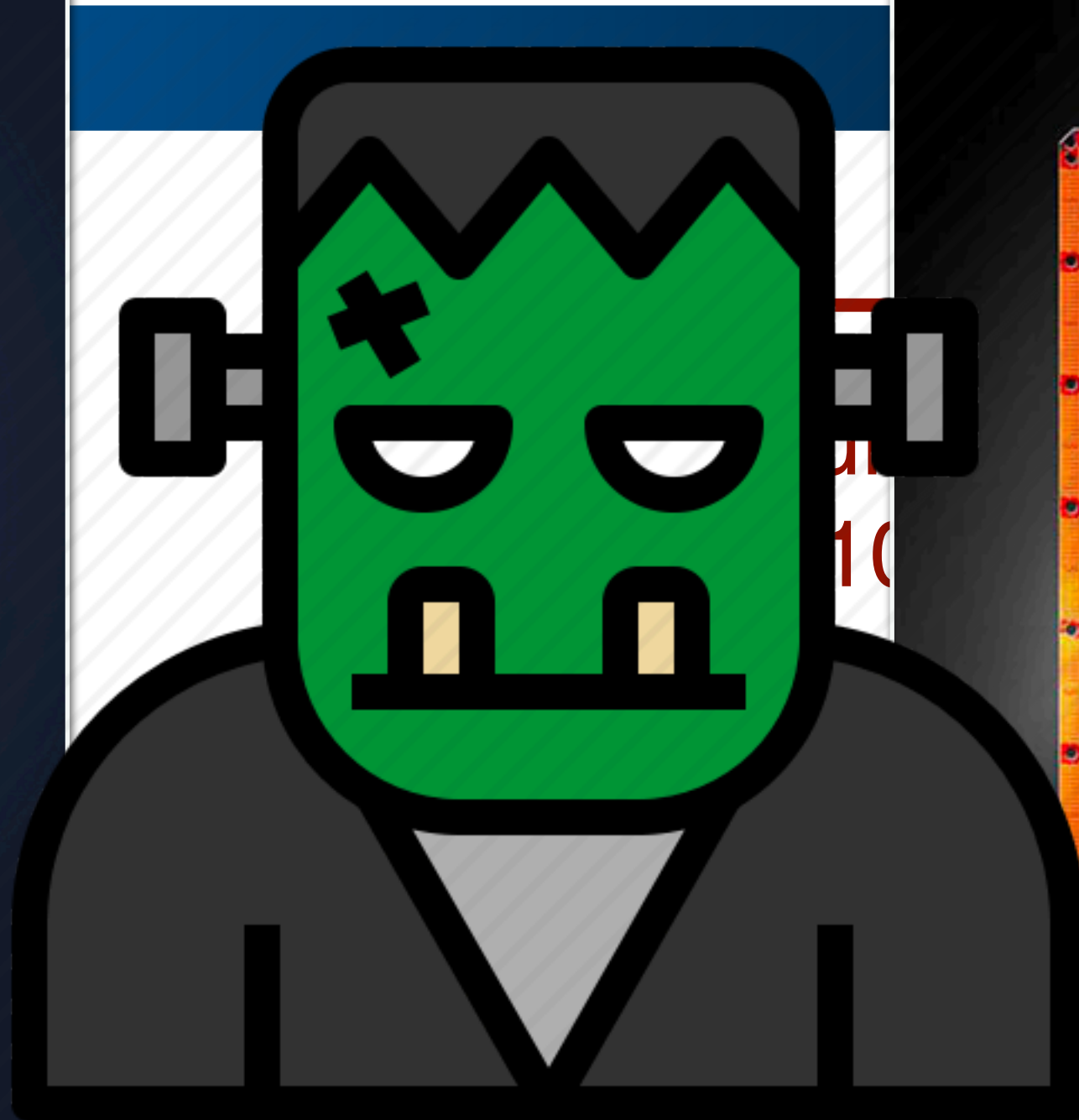
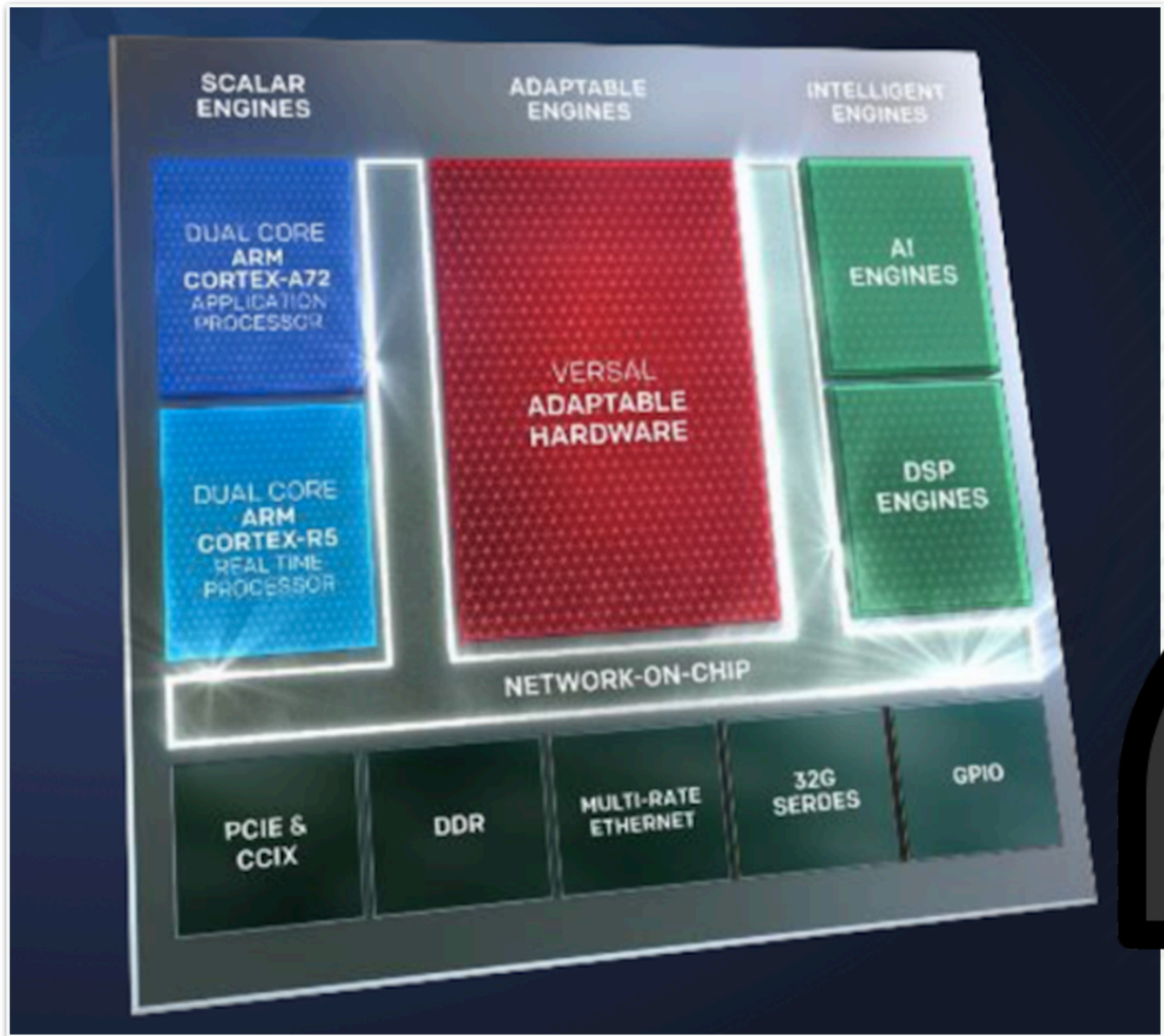
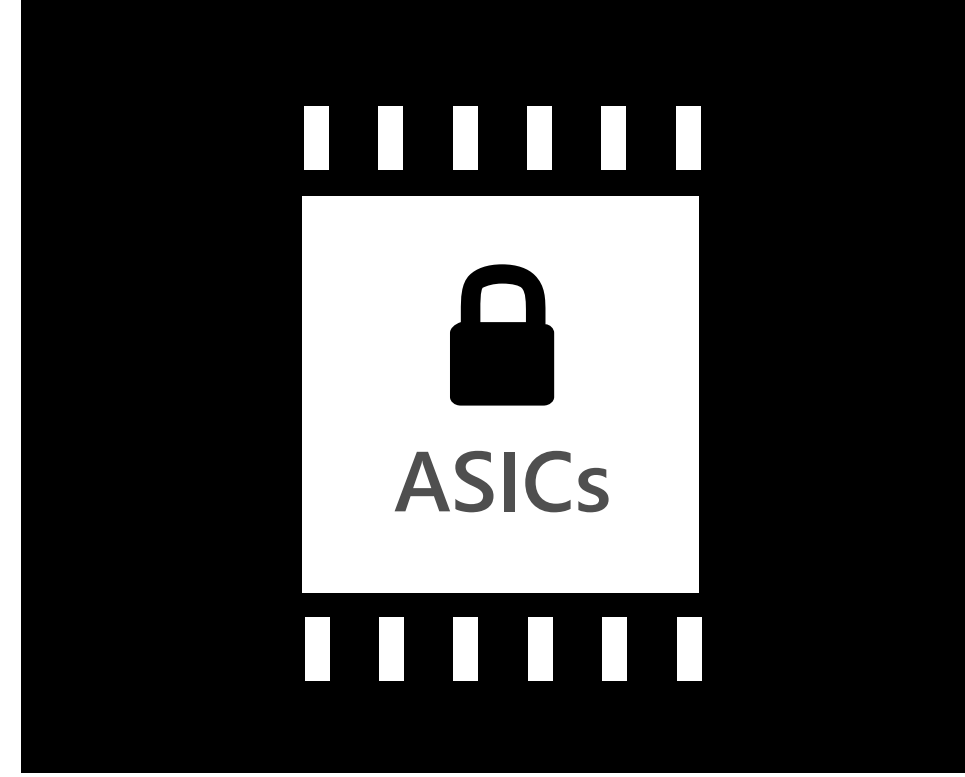
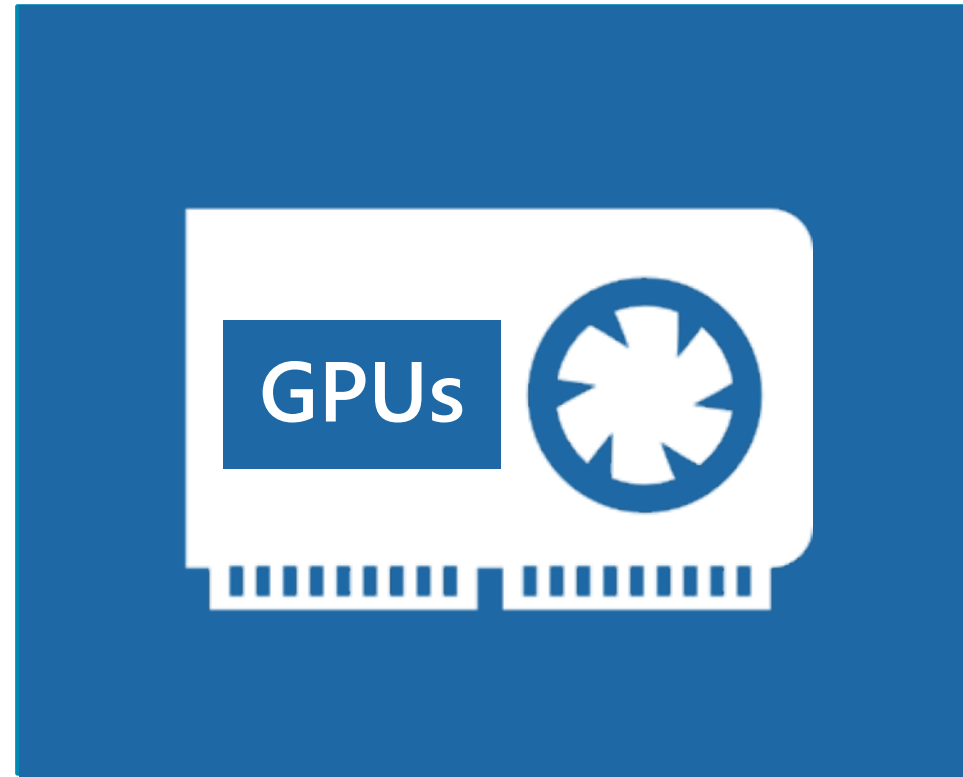
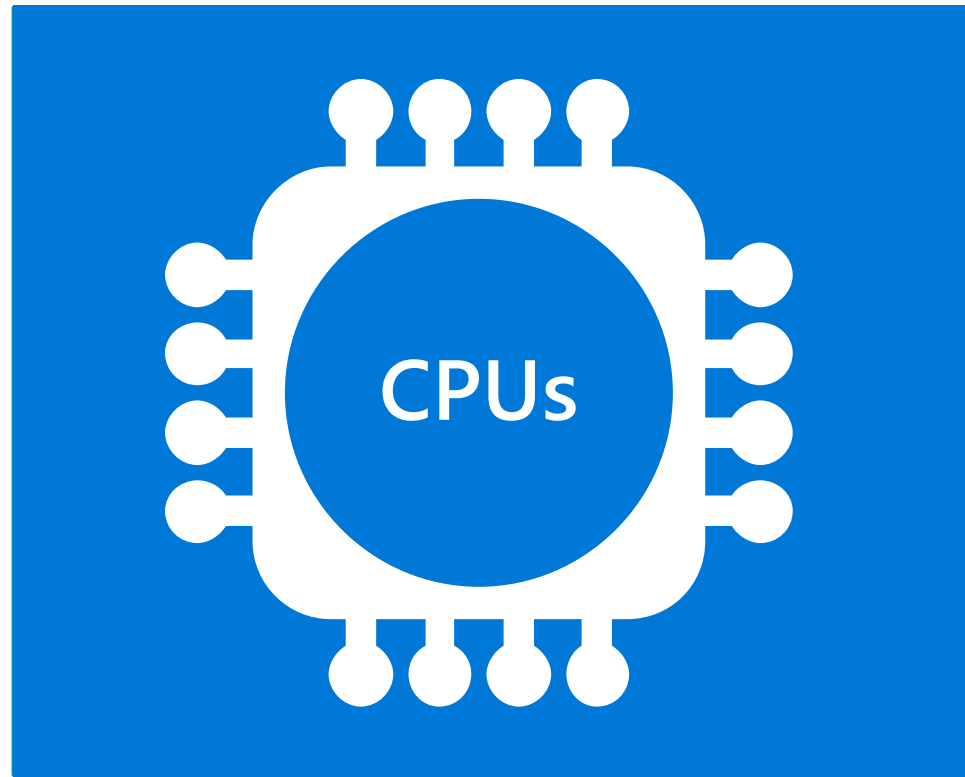


[EIC CDR]

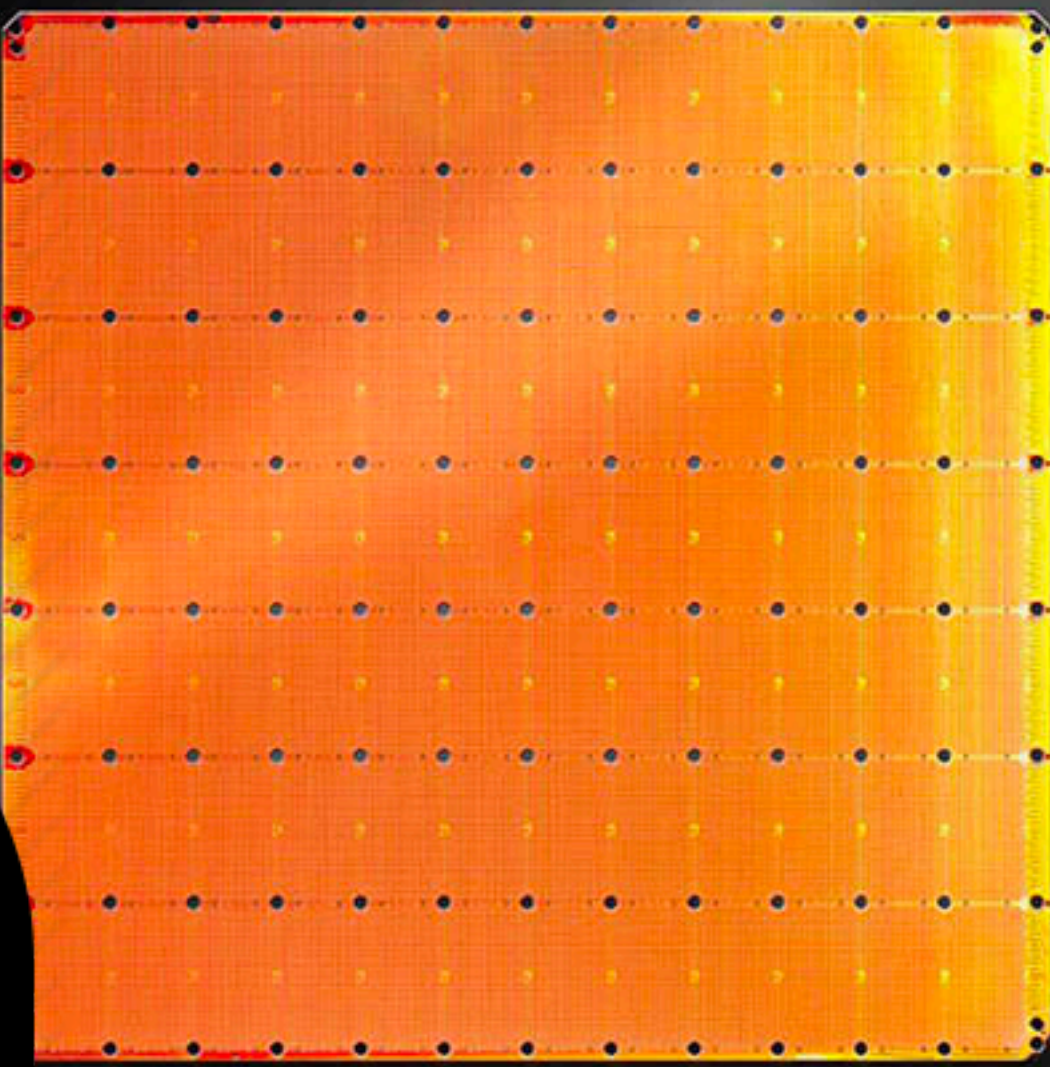


Guidelines:

- > 100 Gbps throughput
- < 1ms computational latency
- < 10W power budget



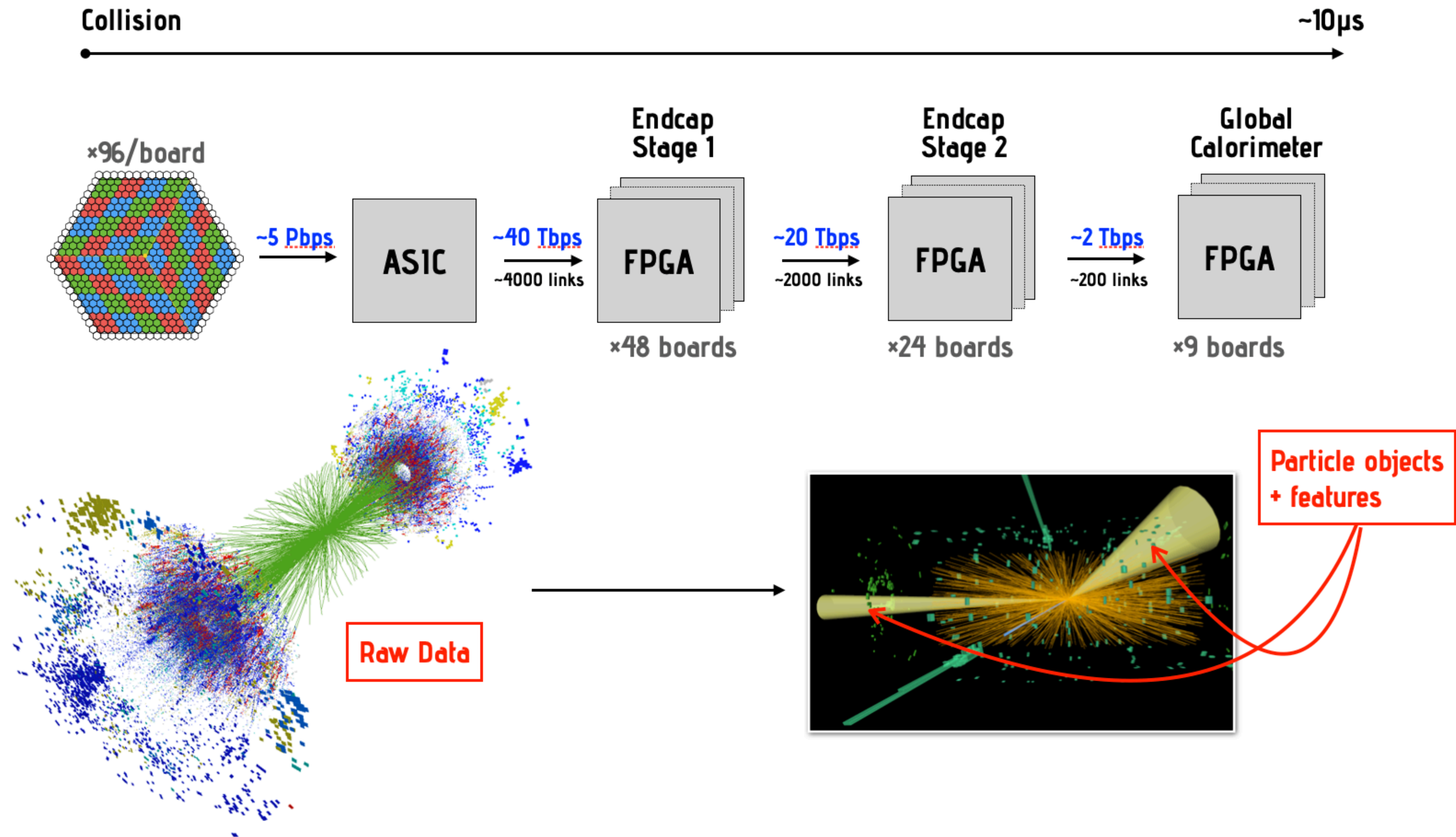
Cerebras Wafer Scale Engine



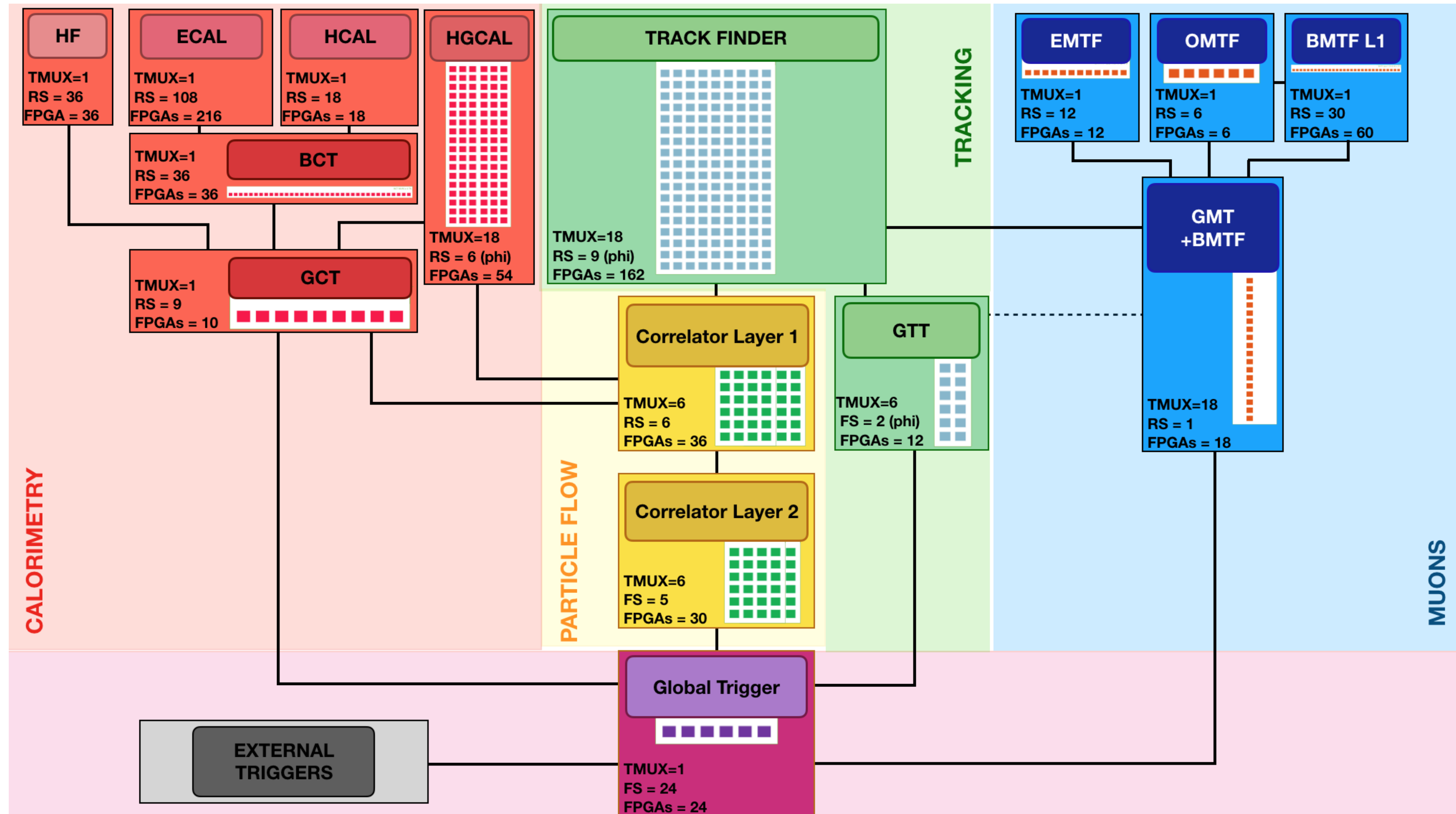
Cerebras WSE
1.2 Trillion Transistors
46,225 mm² Silicon

Largest GPU
21.1 Billion Transistors
815 mm² Silicon

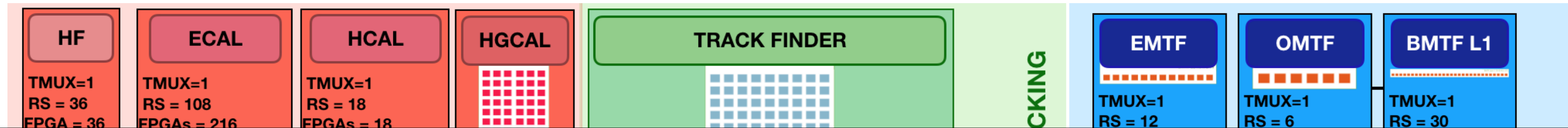
Heterogeneous, hardware-constrained multi-tiered systems



LHC L1 FPGA Trigger as task-based event processing



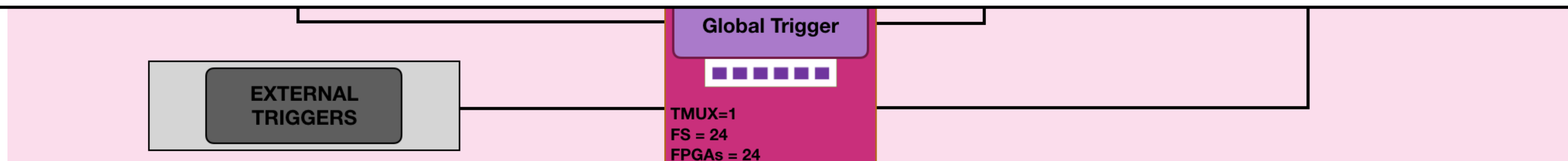
LHC L1 FPGA Trigger as task-based event processing



Each little box is a customized compute microarchitecture for a specific task - a huge job

Can we make ML easily accessible at this stage?

What about when we leave custom embedded systems and go to off-the-shelf computing? How to integrate options beyond CPU? GPU?



Guiding principles

- I have yet to talk about any specific (ML) algorithm or network architecture — there are many approaches, existent and in development
 - Very interesting to see the many different ideas in this workshop
- Designing a **real-time, resource/latency constrained system** adds additional axes of optimization — try to build an adaptable, flexible, scalable system
 - Accessible at each layer of data processing
 - ML provides powerful data reduction techniques, has the potential to account for unknown unknowns

Efficient ML in trigger

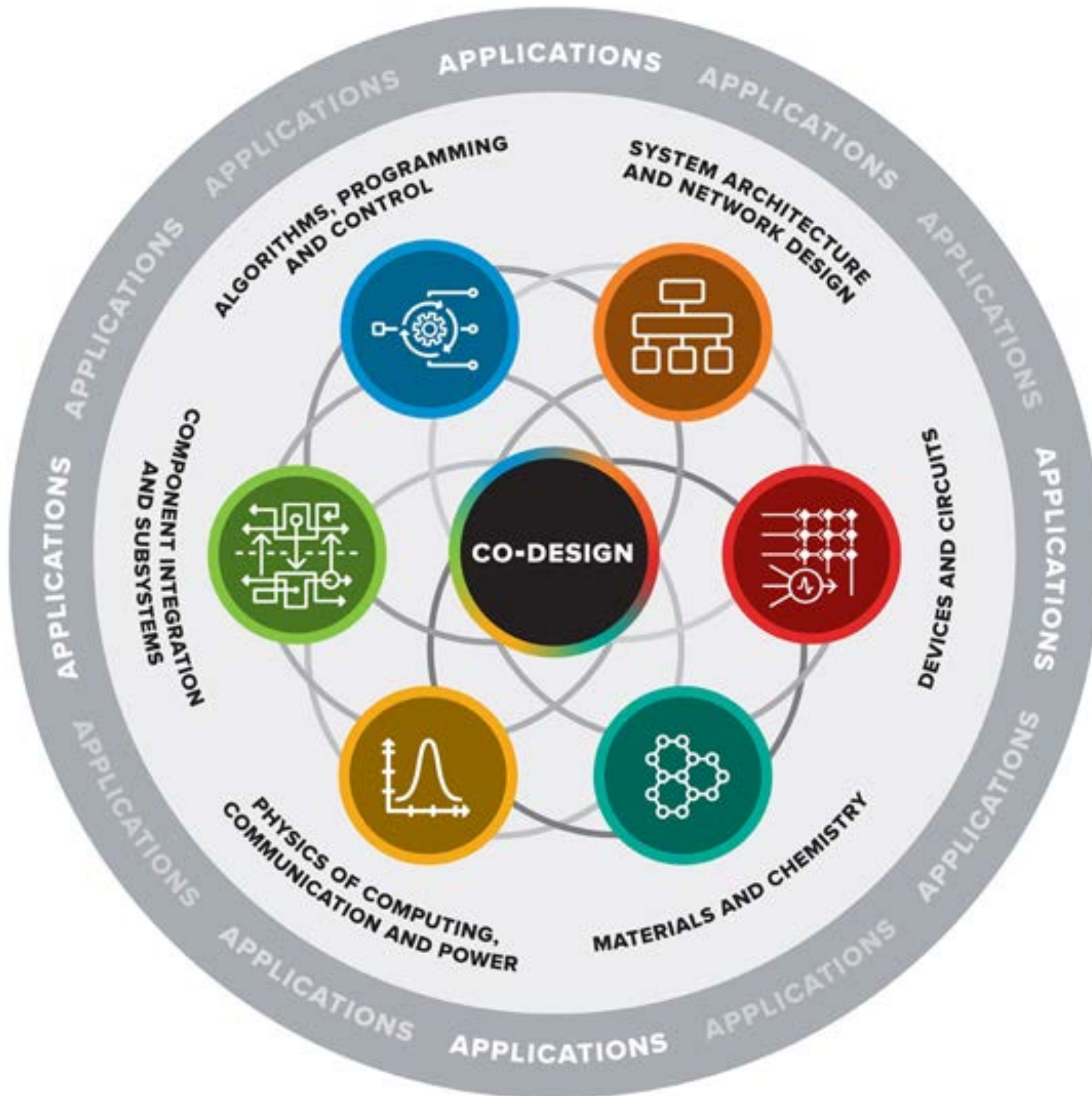
- techniques and examples**

{algorithms, implementations, tool flows for}

^

Efficient ML in trigger

— techniques and examples



Algorithms:

e.g. AlexNet to SqueezeNet/OnceForAll, NAS

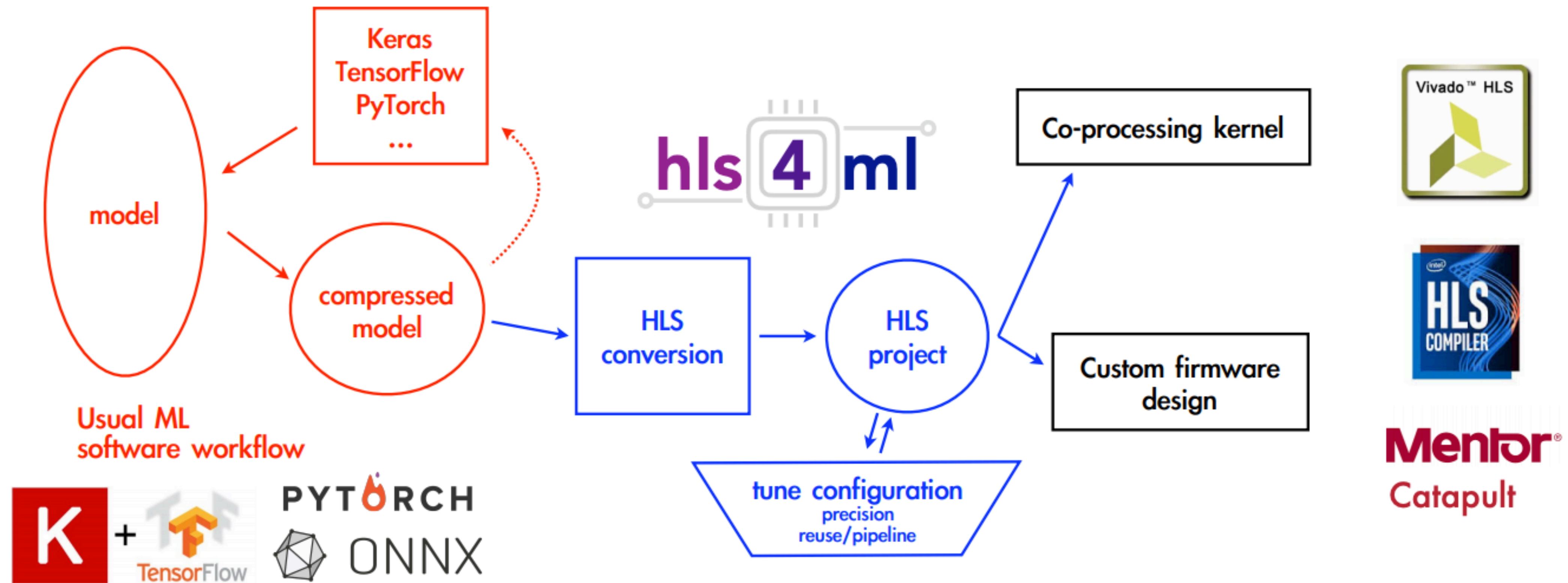
Implementations:

e.g. Quantization, Pruning, Dataflow

Tool flows:

e.g. IRs, synthesis, EDA

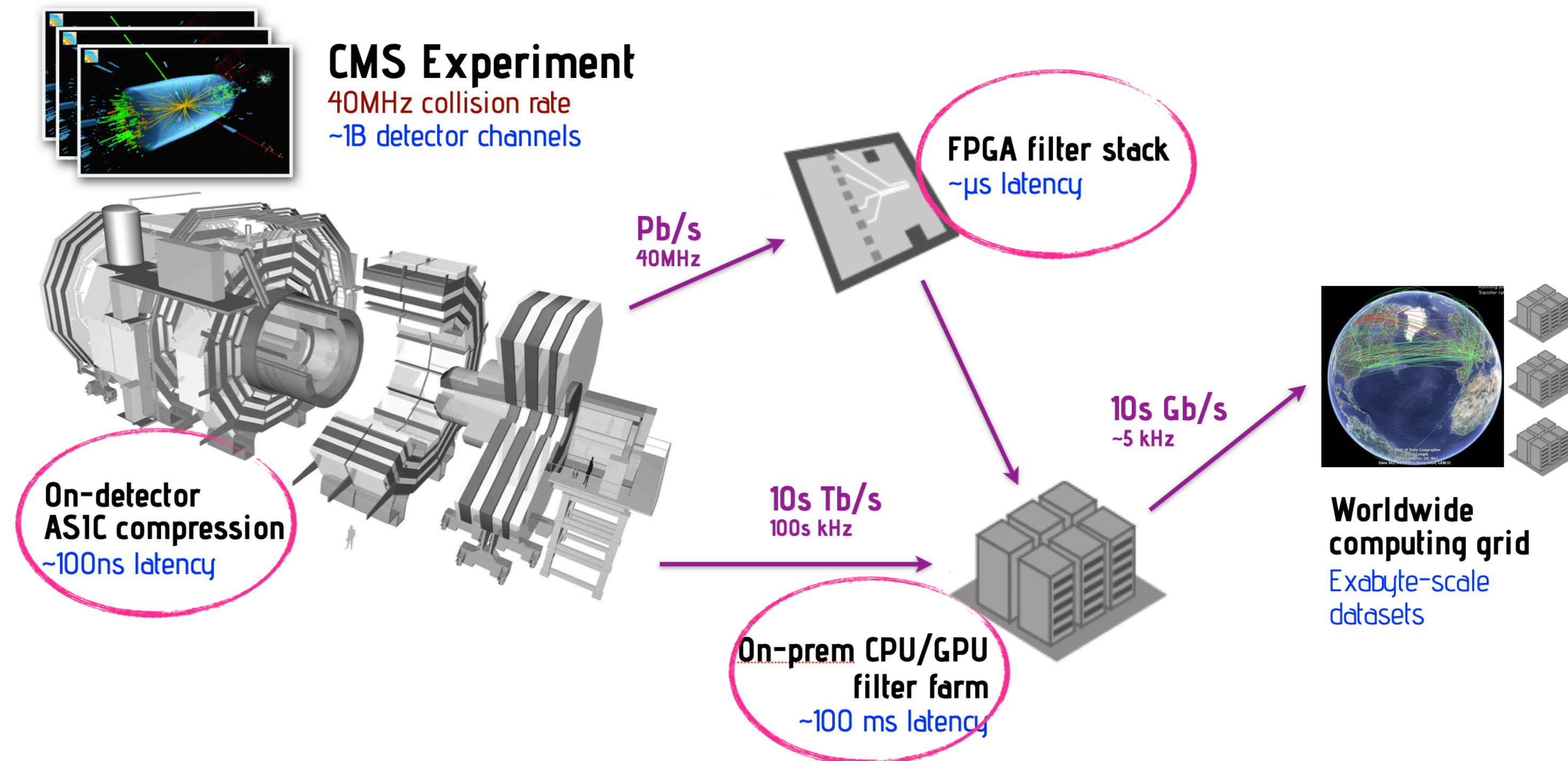
hls4ml: a codesign workflow!

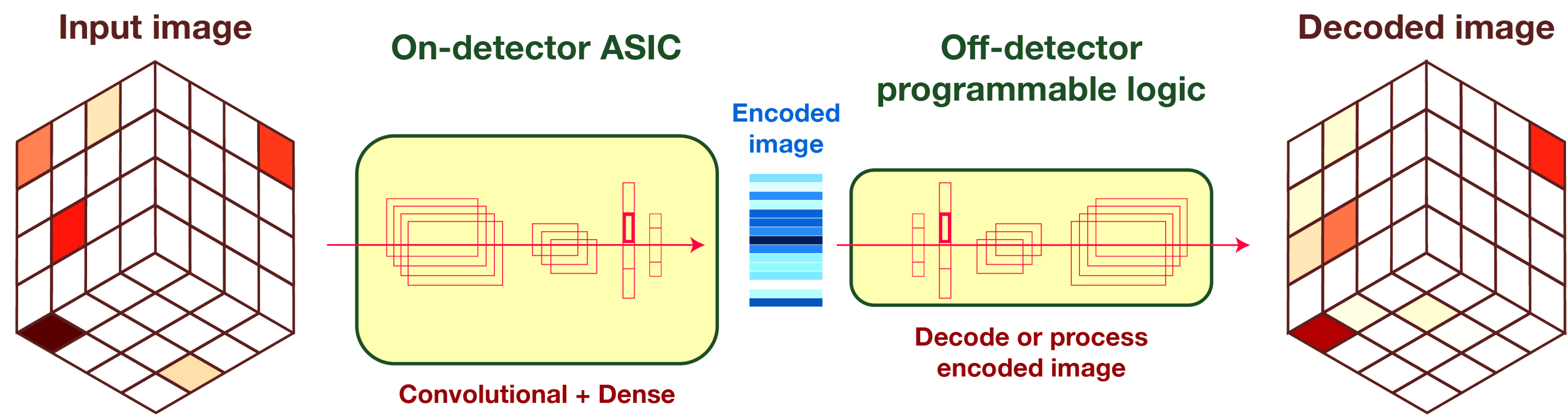


See Dylan Rankin's talk!

Examples

- The fronttest end of the detector – reconfigurable ASIC data compression
- Level-1 FPGA trigger deployment
- Efficient deployment of coprocessors in trigger





- **Enable** more computationally complex compression algorithms
- **Customize** the compression algorithm per sensor location
- **Adapt** the algorithm for changing conditions, new ideas

Fixed algorithm architecture, but allow weights to be reconfigurable

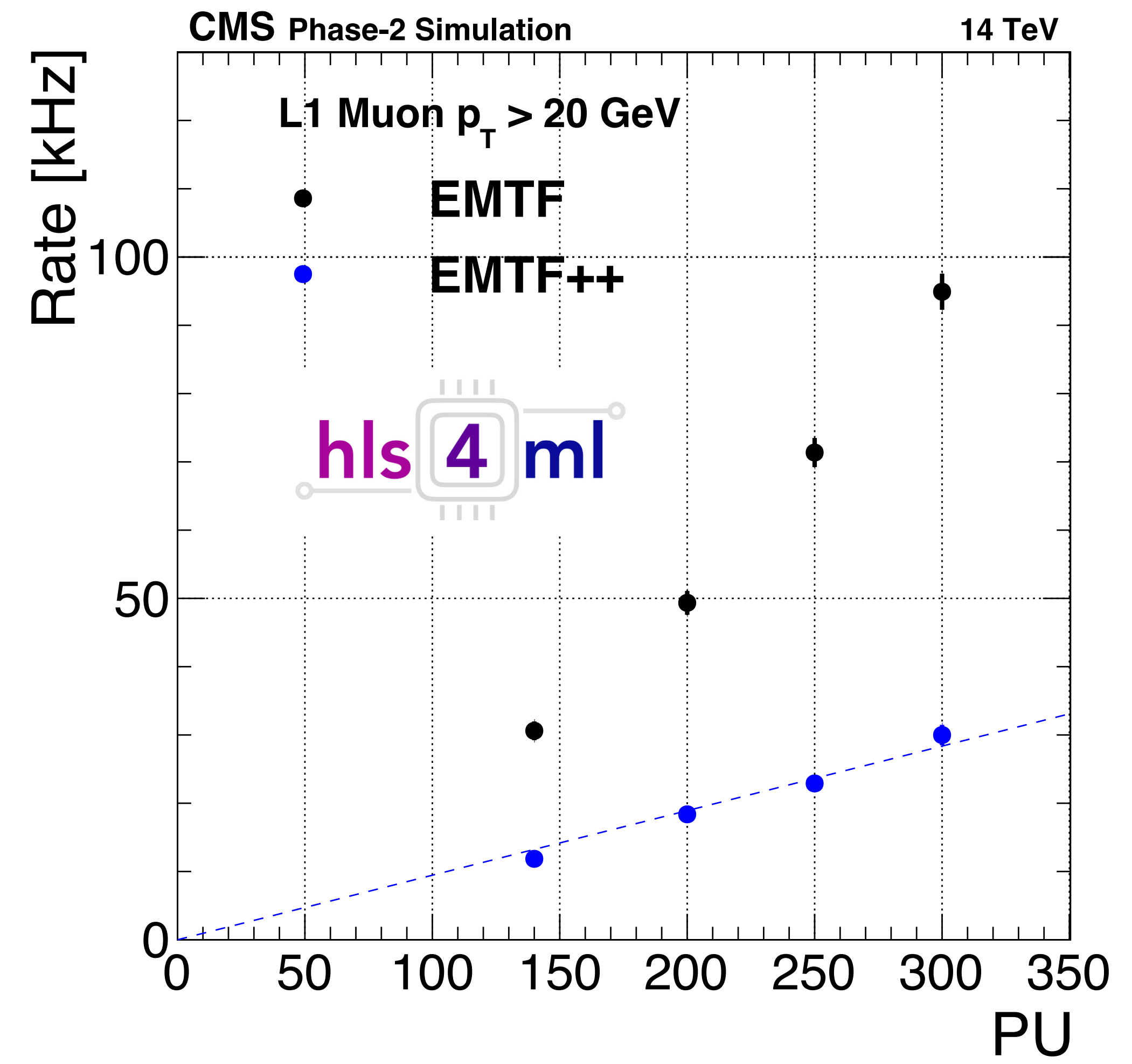
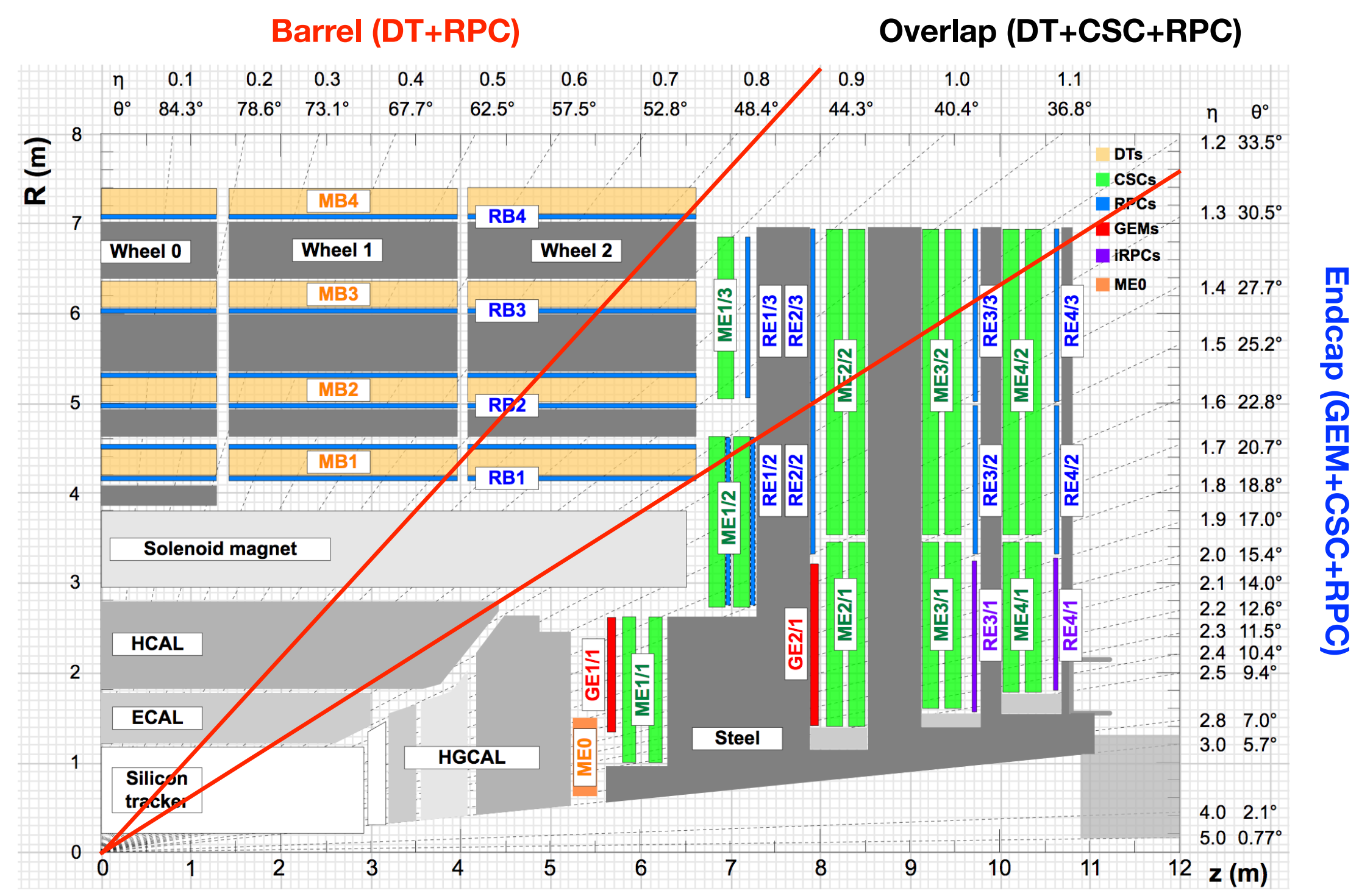
Requirements	
Rate	40 MHz
Total ionizing dose	200 Mrad
High energy hadron flux	$1 \times 10^7 \text{ cm}^2/\text{s}$

Metric	Simulation	Target
Power	48 mW	<100 mW

See Farah Fahim's talk!

Open questions, how to optimally build latent space, use representation downstream?

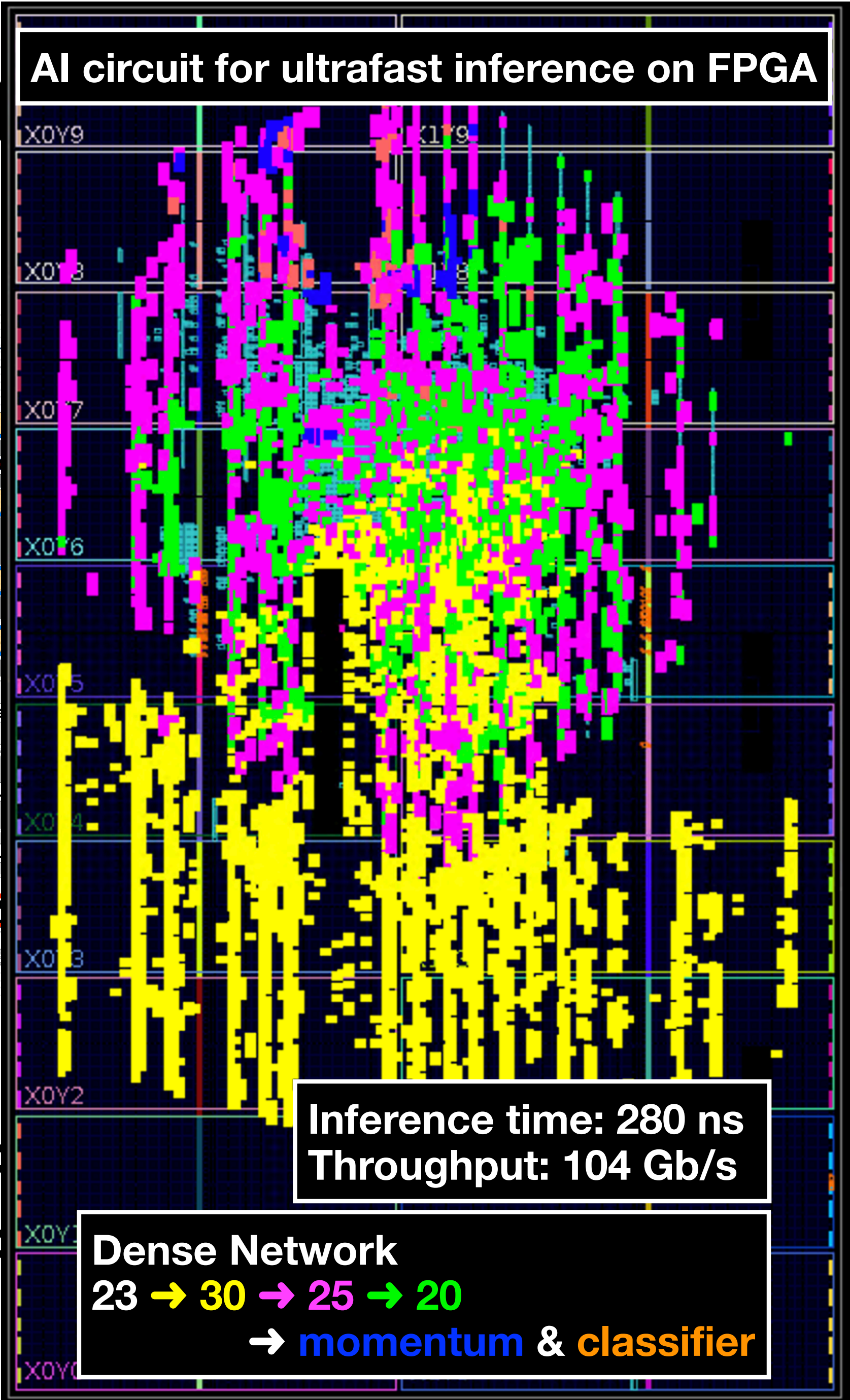
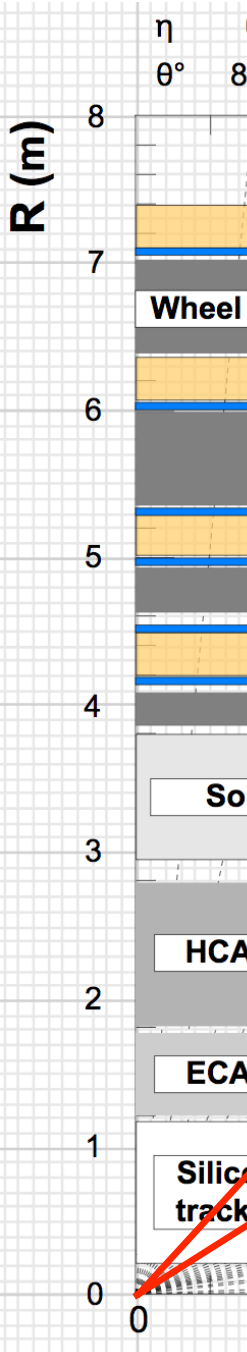
Case study: muon trigger upgrade



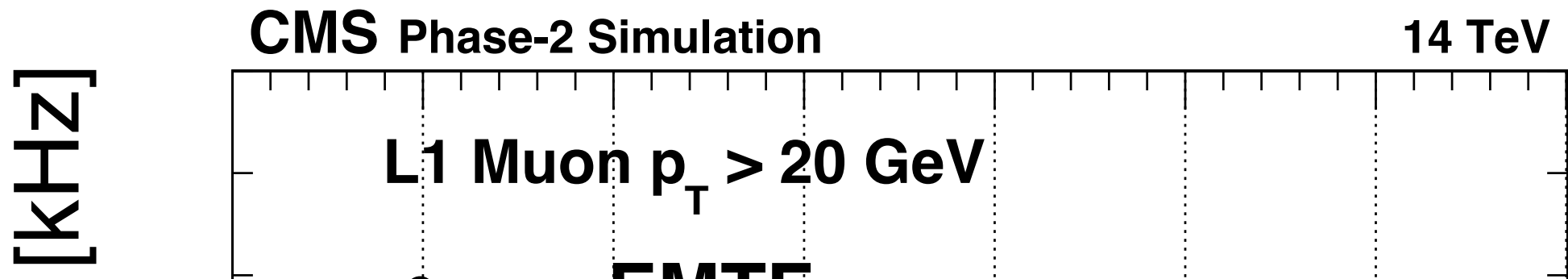
EMTF = BDT (external memory)
 EMTF++ = fully connected NN
 ~3x reduction in the trigger rate for neural network!

Case

trigger upgrade

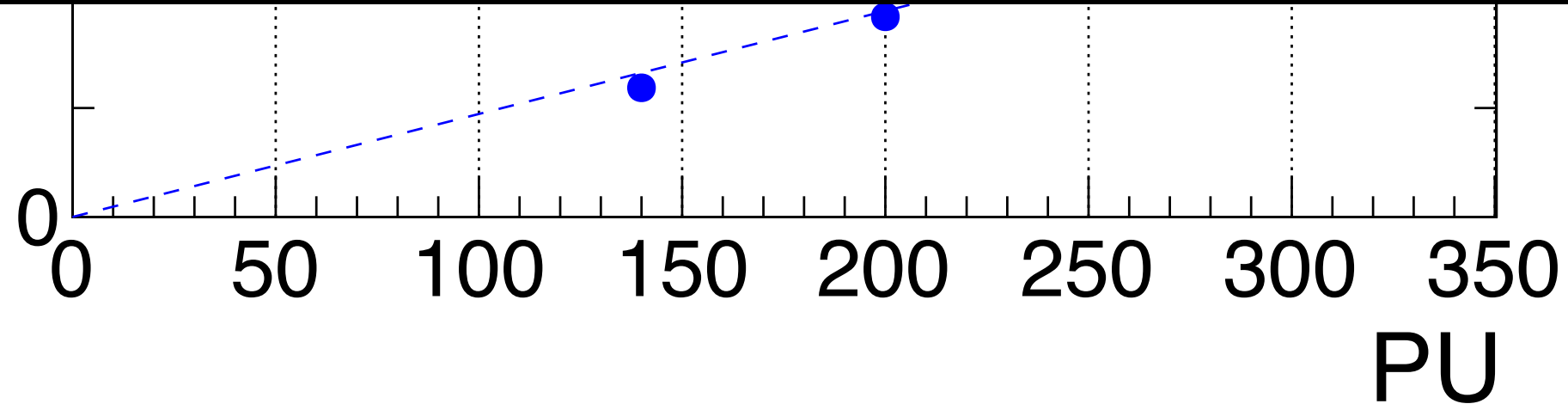


Endcap (GEM+CSC+RPC)



Open questions, how to train, monitor, and calibrate such online algorithms? Can/should we automate updates?

More discussion on this later...



EMTP
EMTP
~3x

r neural network!

Compute acceleration



Algorithm

Domain

ML

Integration

as a Service
(aaS)

direct
connect

Hardware

CPU

GPU

FPGA

ASIC

...

How to accommodate an unknown number of algorithms on an unknown hardware platform?

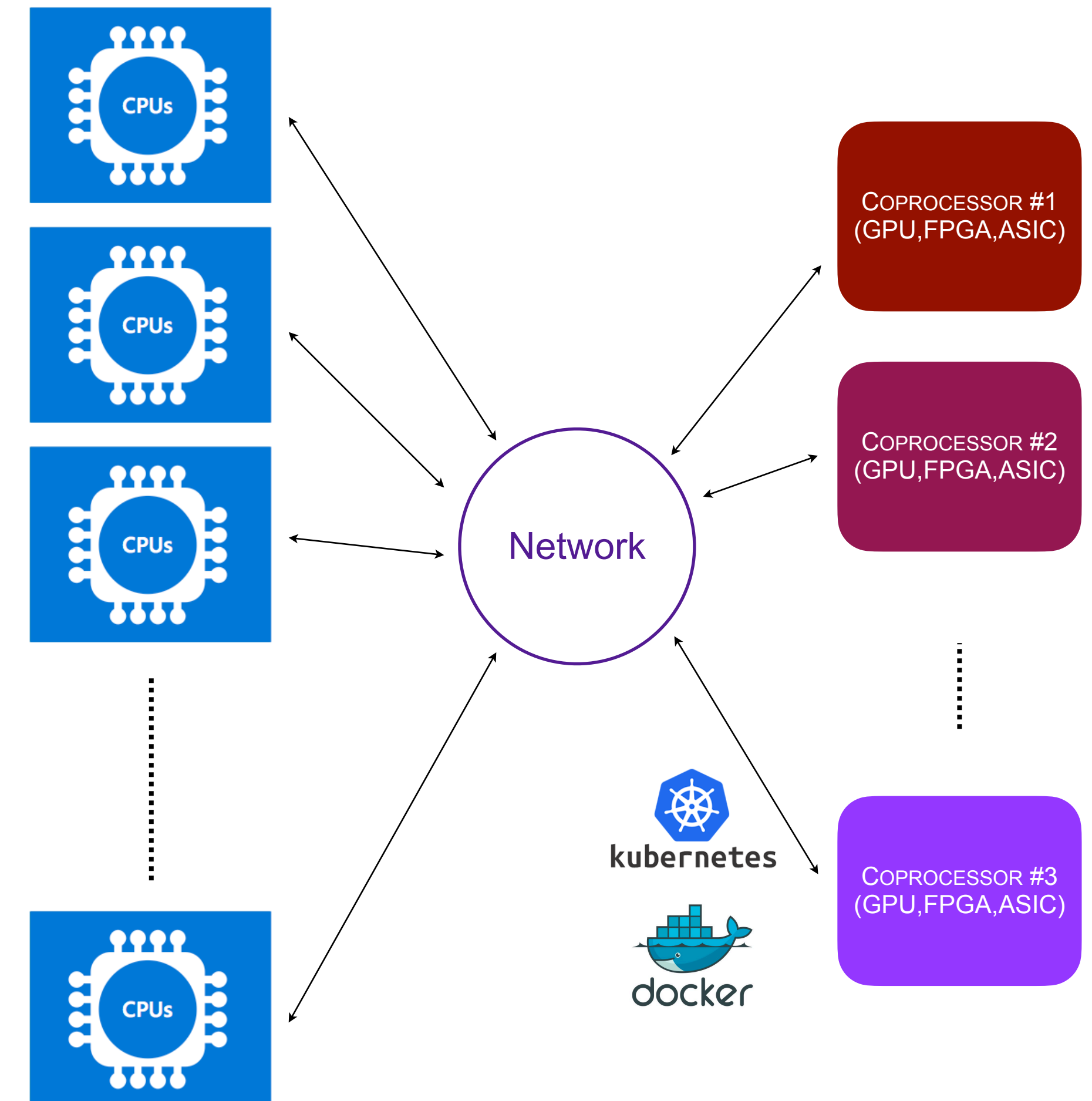
SONIC

Services for **O**ptimized **N**etwork **I**nference on **C**oprocessors

Flexible - optimize the hardware based on task; no need to support many ML frameworks in experiment software

Adaptable - right-size the system to the task, you choose the number of coprocessors based on computing needs

Scalable - coprocessor need not be co-located next to existing CPU infrastructure; common software framework



$N_{\text{CPU}} \neq N_{\text{coprocessor}}$

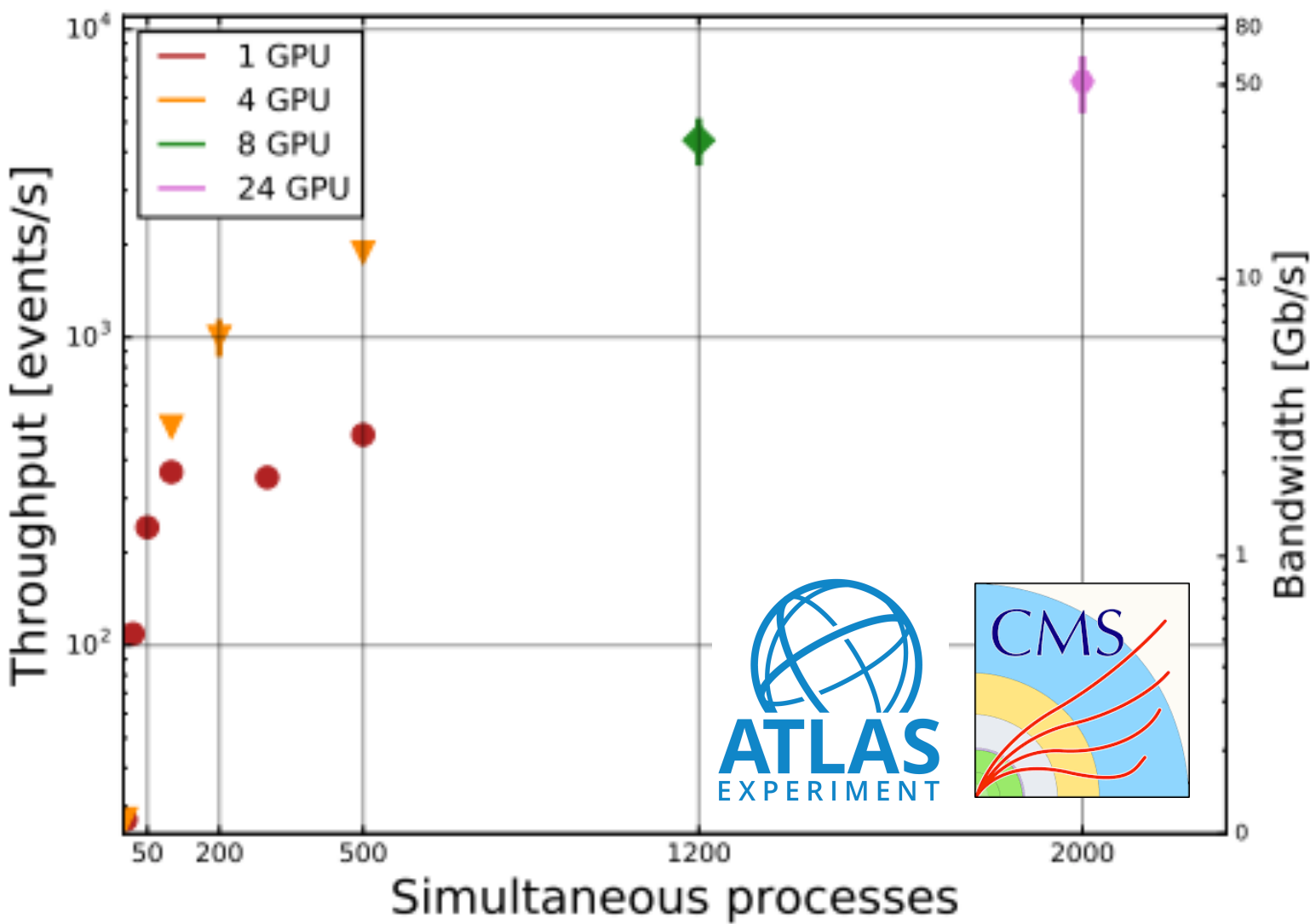
Results

Demonstrated significant and efficient acceleration of LHC/ProtoDUNE tasks

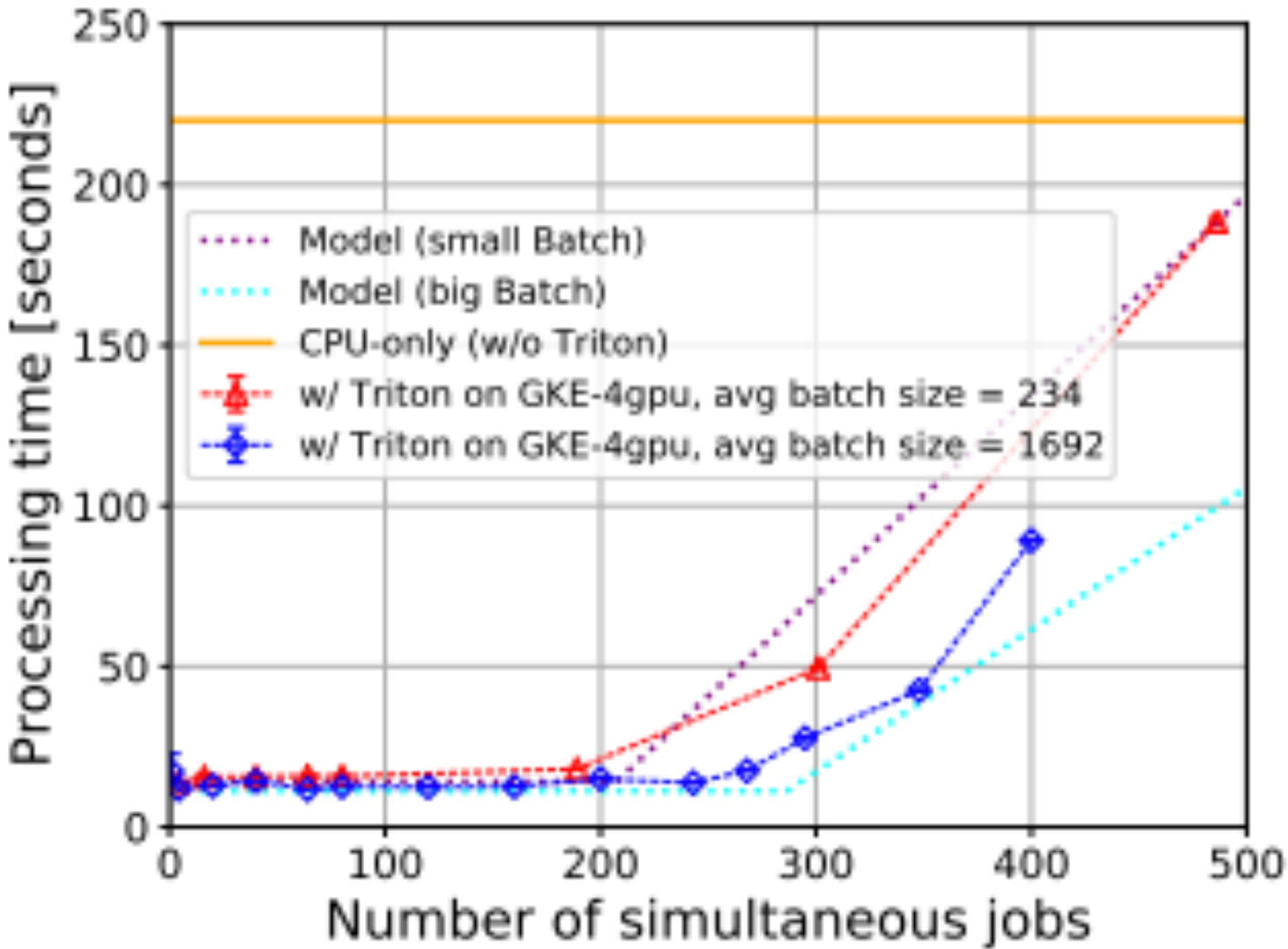
Broad range of tasks — cluster calibration, jet tagging, cosmics ID, Graph NNs

Deployed on-premises, in the cloud, and at HPC – exploring all types of new hardware (FPGA, GPU, TPU, ...)

References:
[arXiv:1904.08986](#)
[arXiv:2007.10359](#)
[arXiv:2009.04509](#)
[arXiv:2010.08556](#)

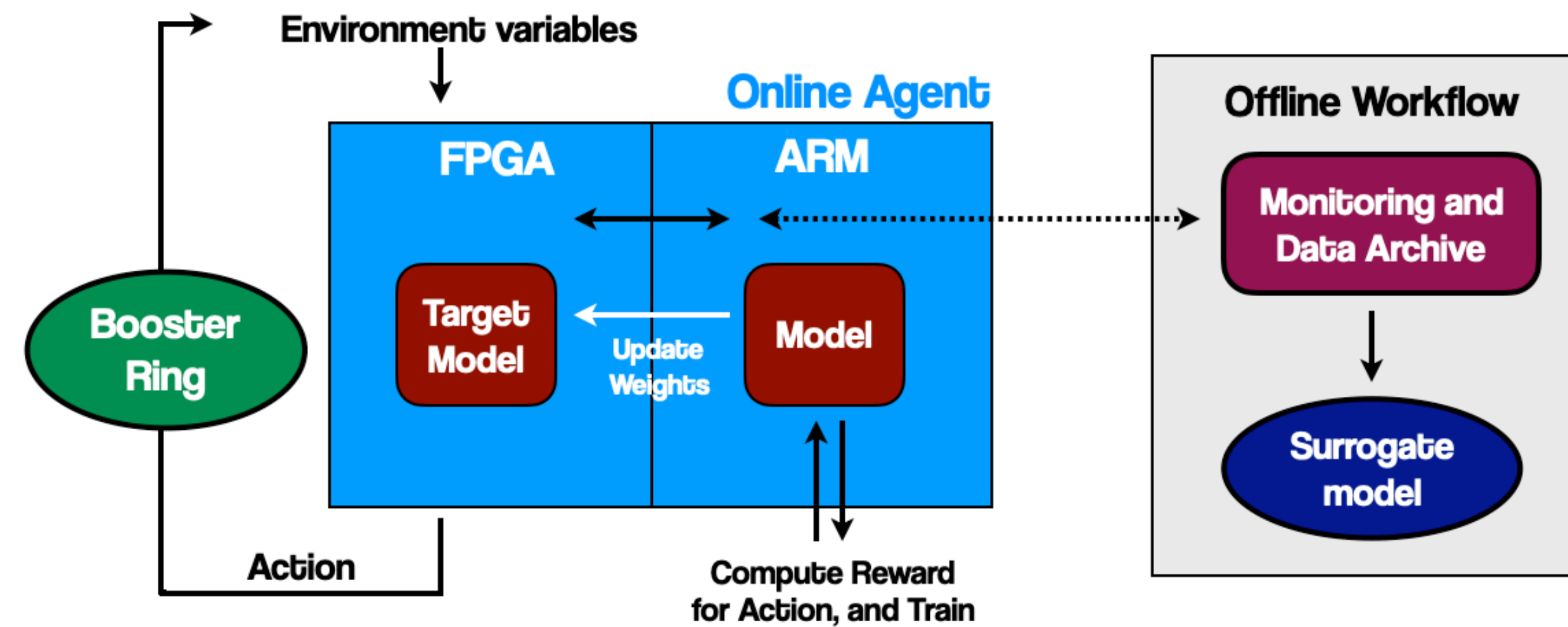


	Wall time (s)		
	ML module	non-ML modules	Total
CPU only	220	110	330
CPU + GPUaaS	13	110	123

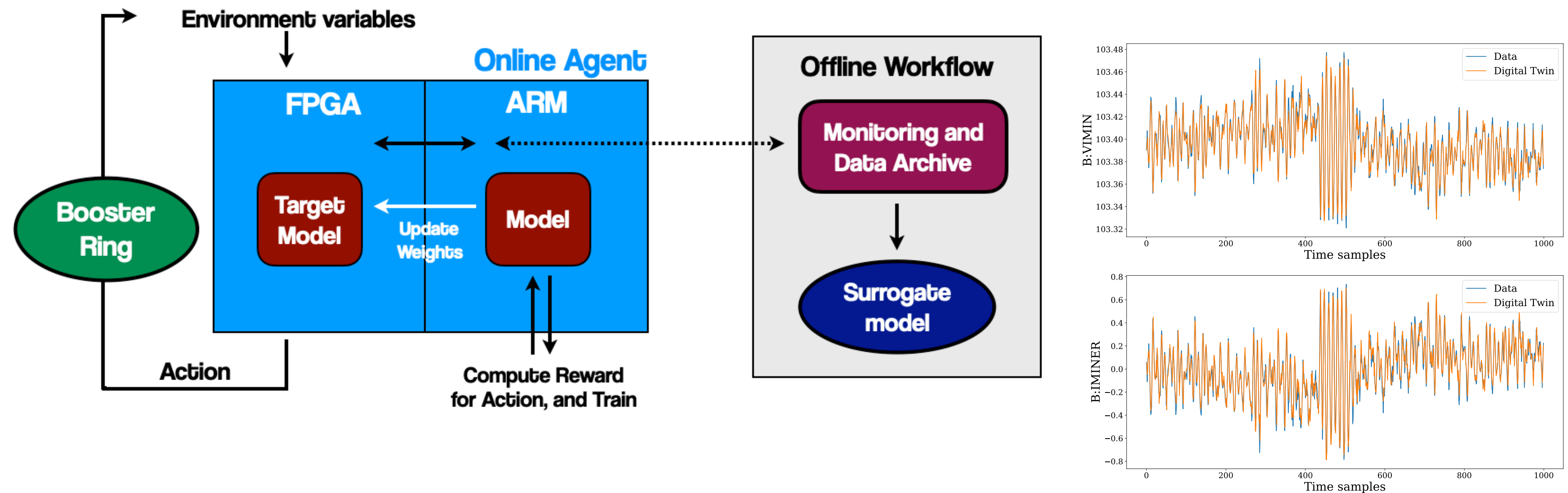


Open-ended thoughts and future challenges

Continuous learning setup



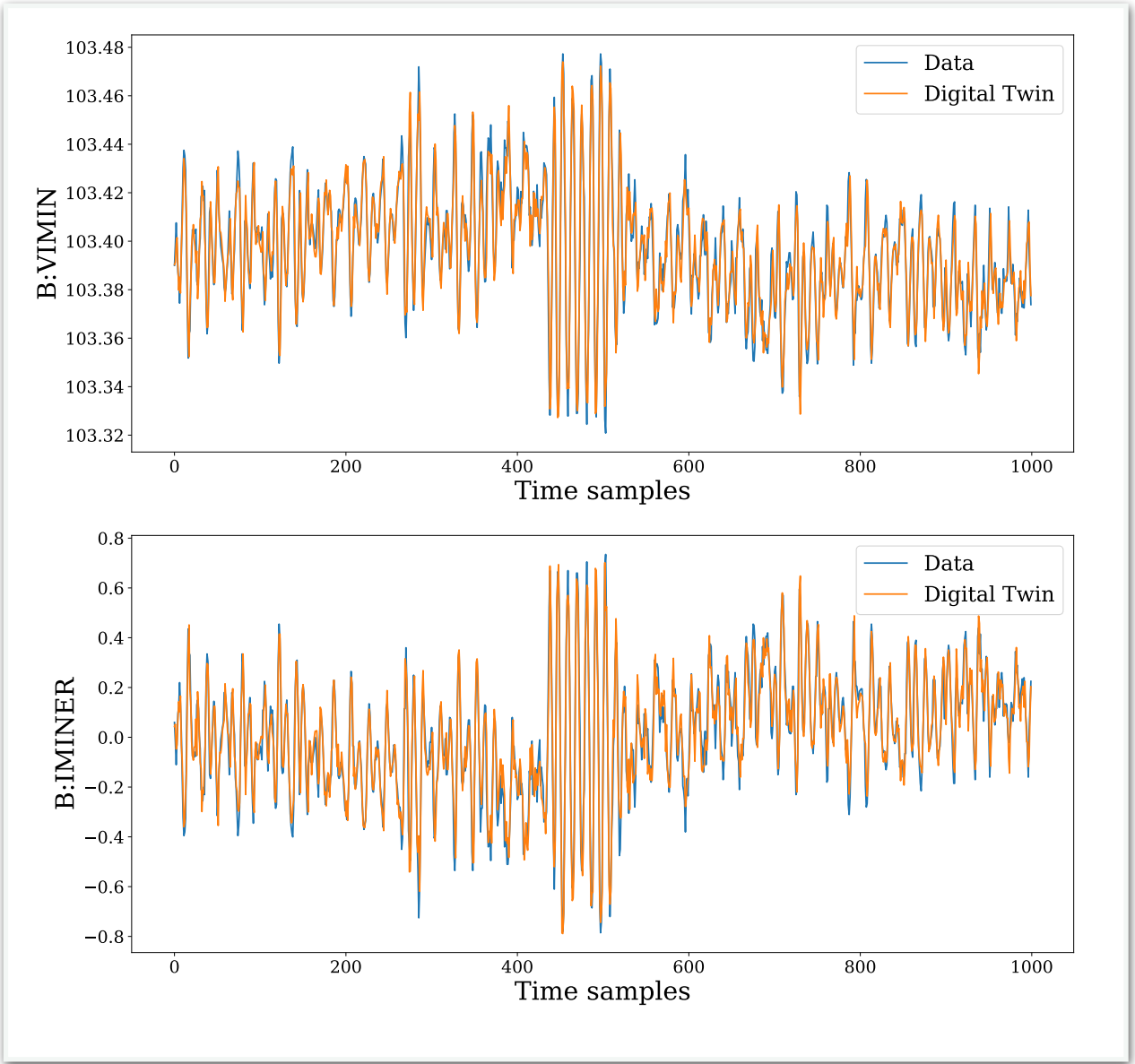
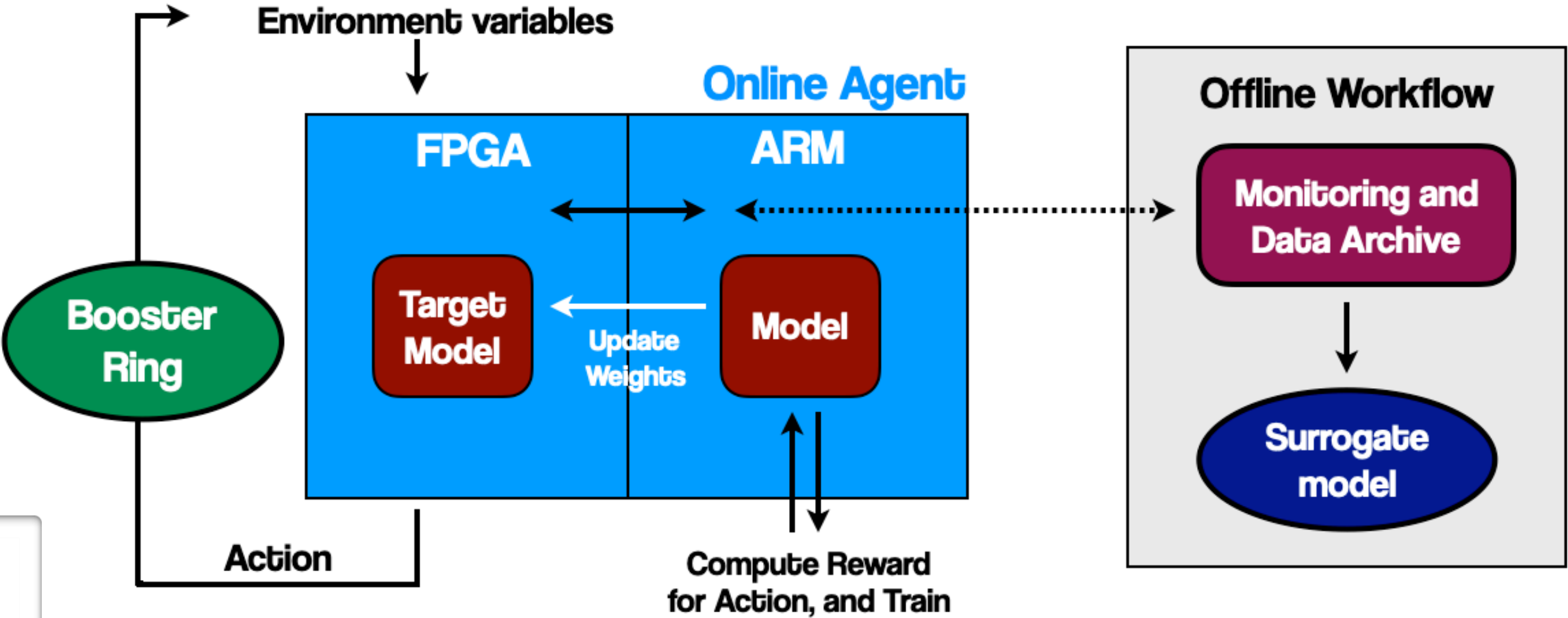
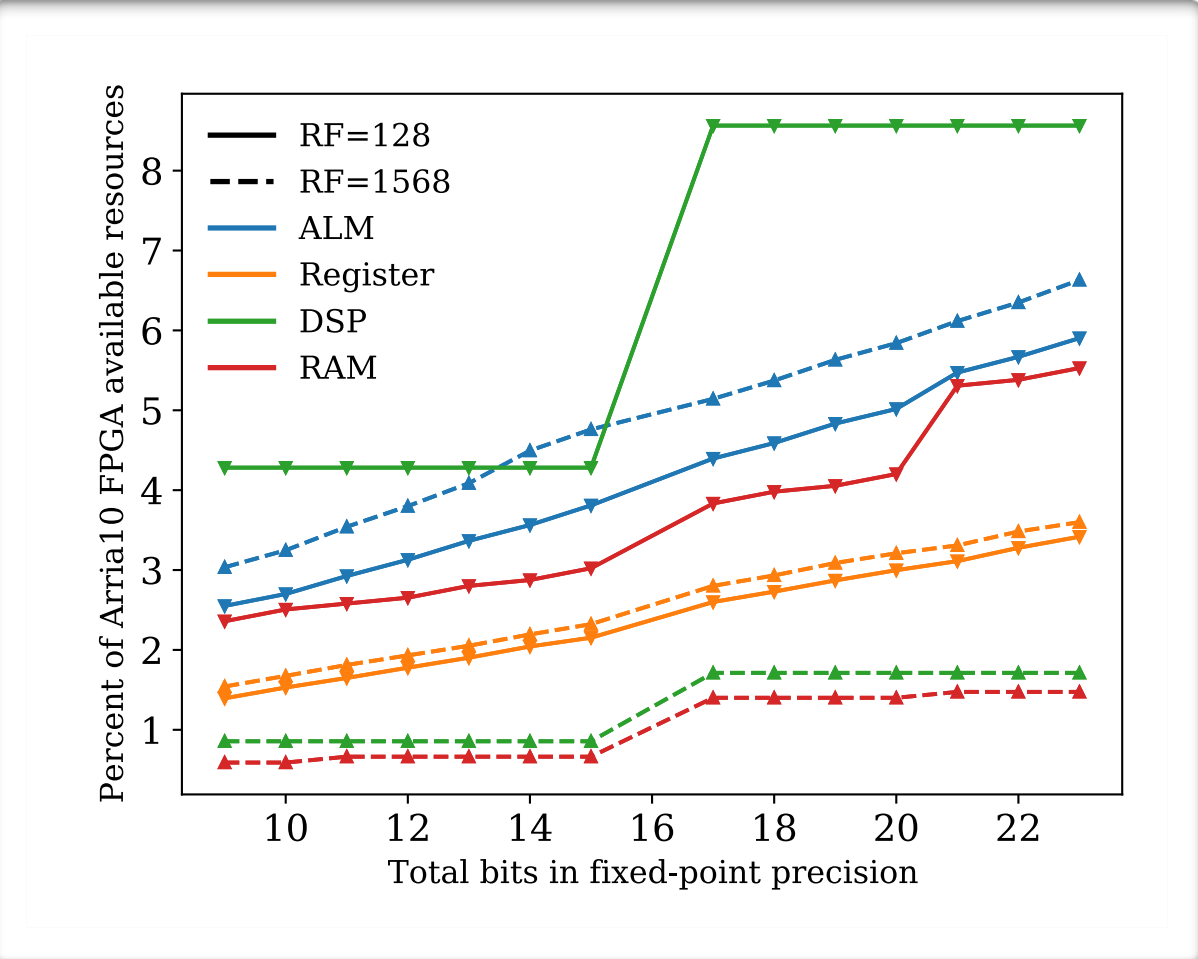
Continuous learning setup



LSTM-based network using
Deep Q-learning framework
for surrogate model to mimic
behavior of Booster

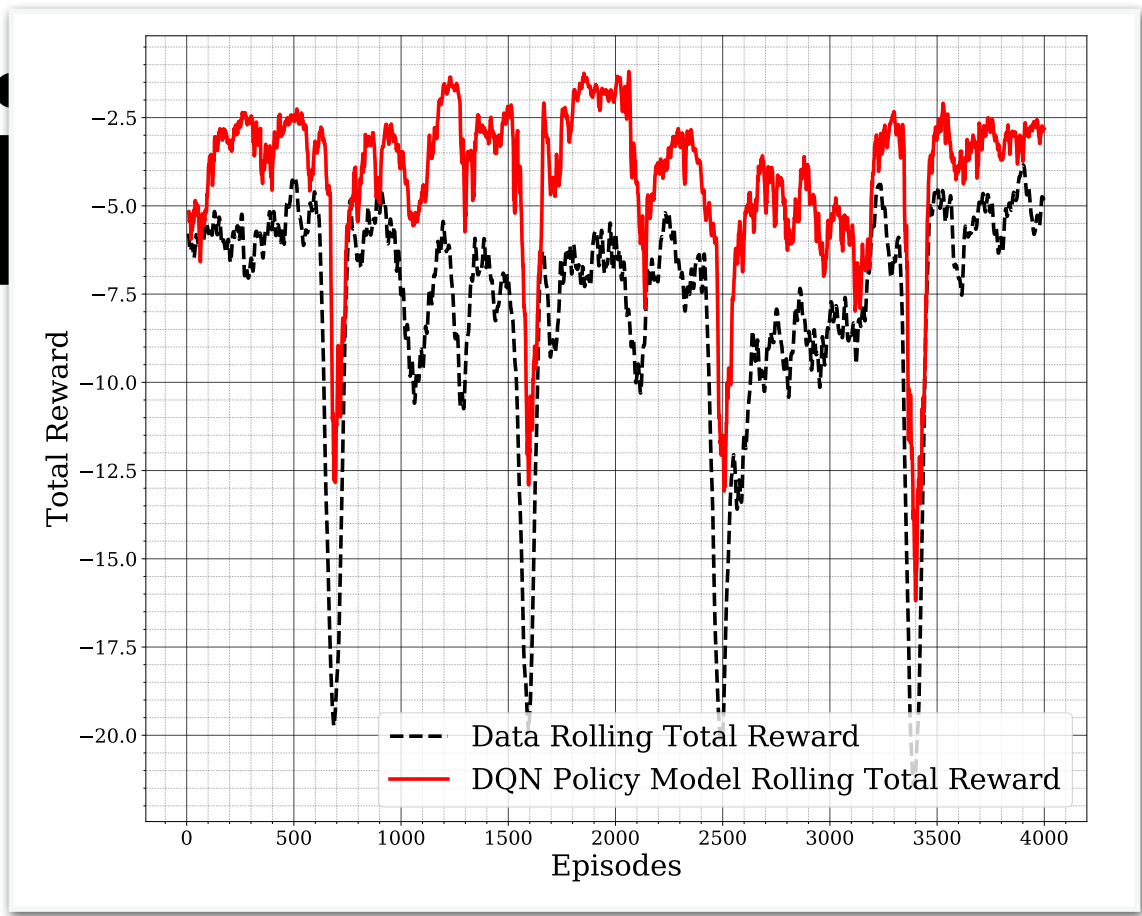
Continuous learning setup

DNN-based ensemble
model for RL online agent



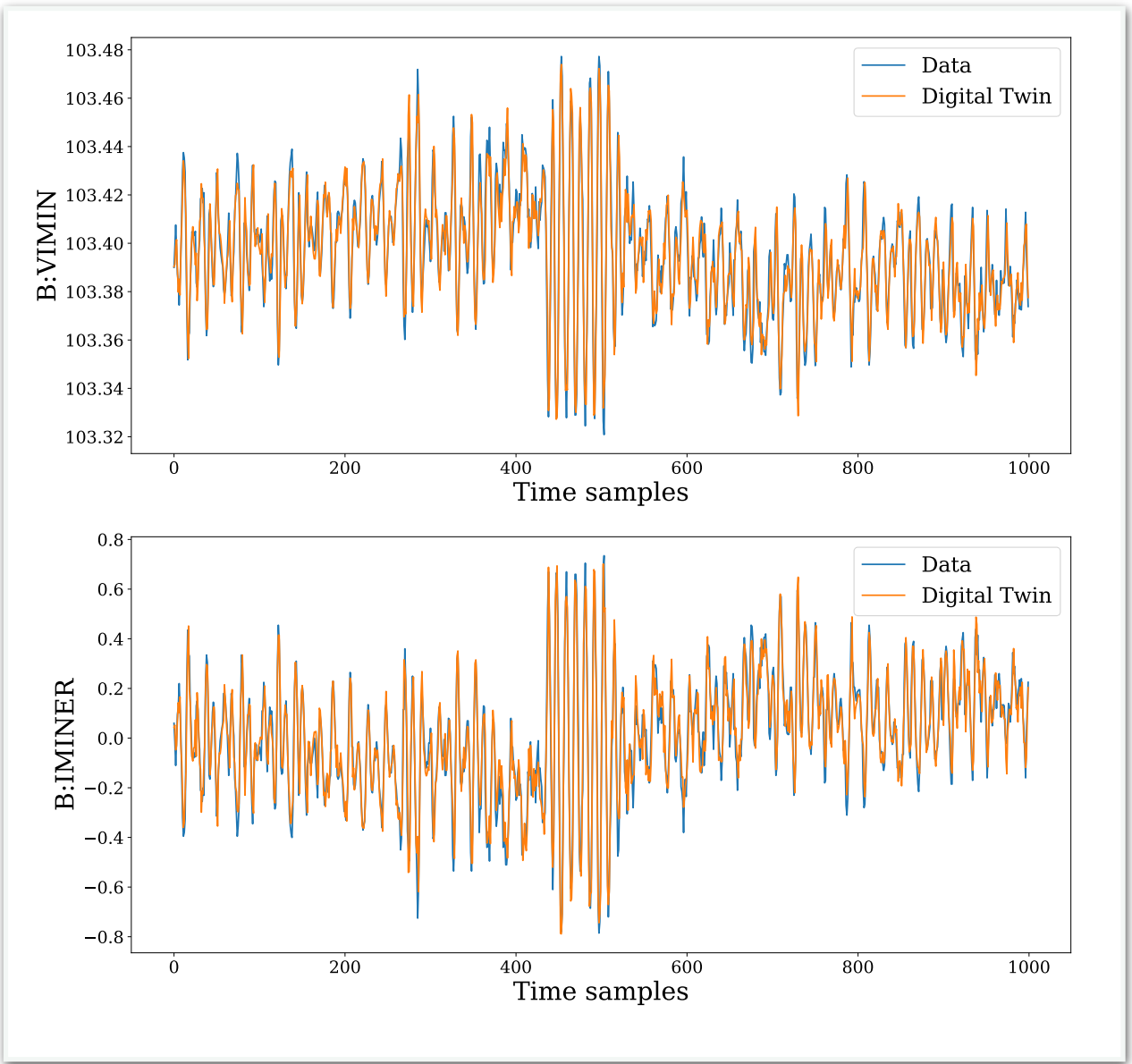
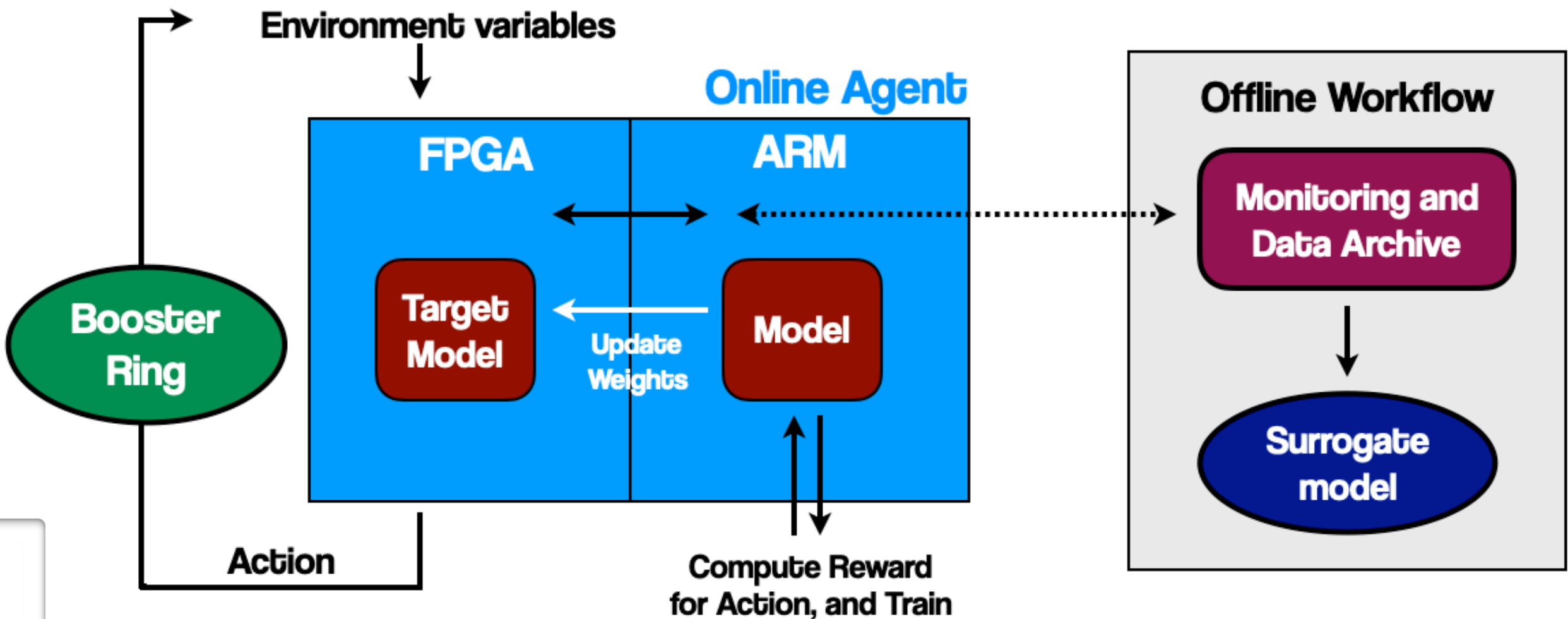
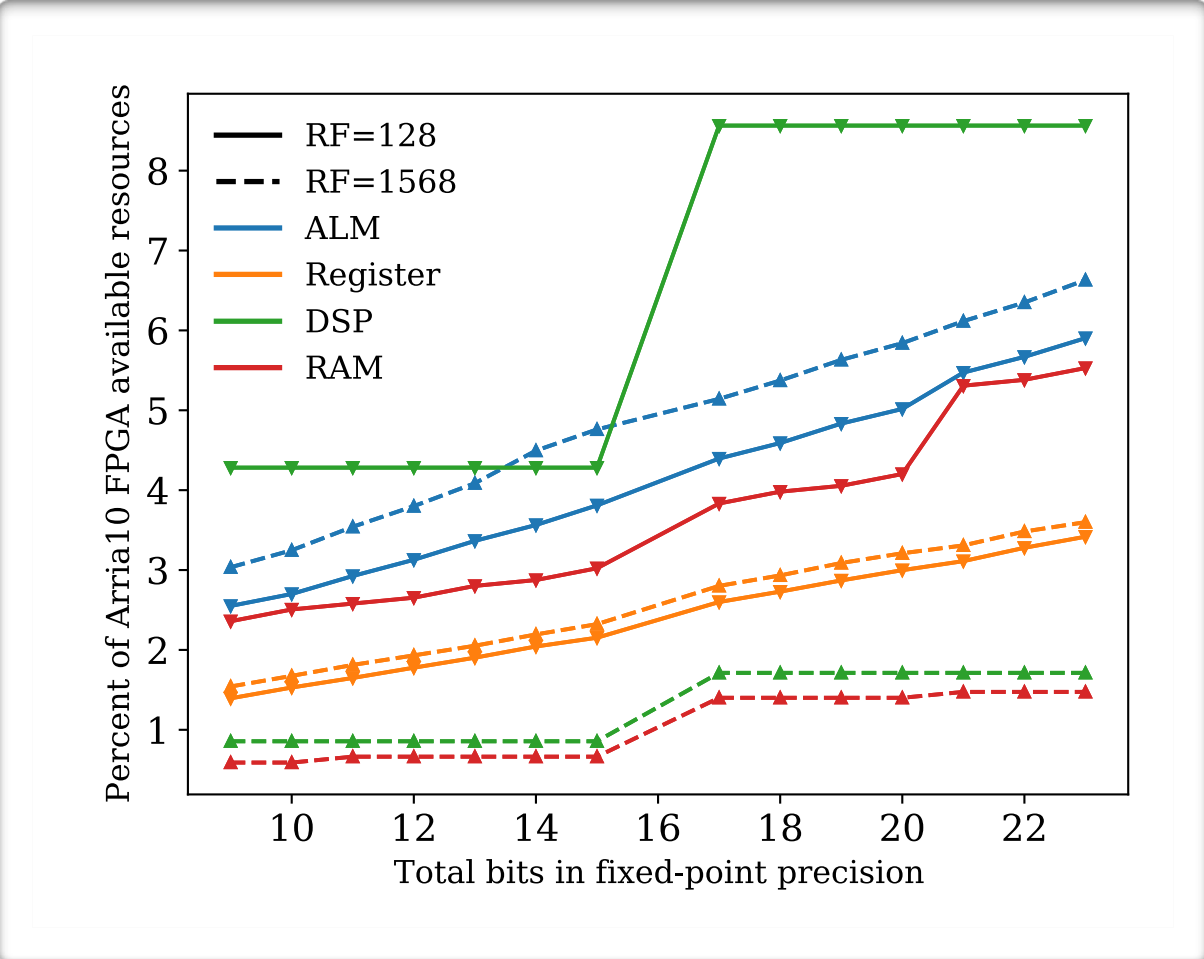
LSTM-based network using
Deep Q-learning framework
for surrogate model to mimic
behavior of Booster

Continuous learning



Reduced beam losses predicted from RL approach

DNN-based ensemble model for RL online agent



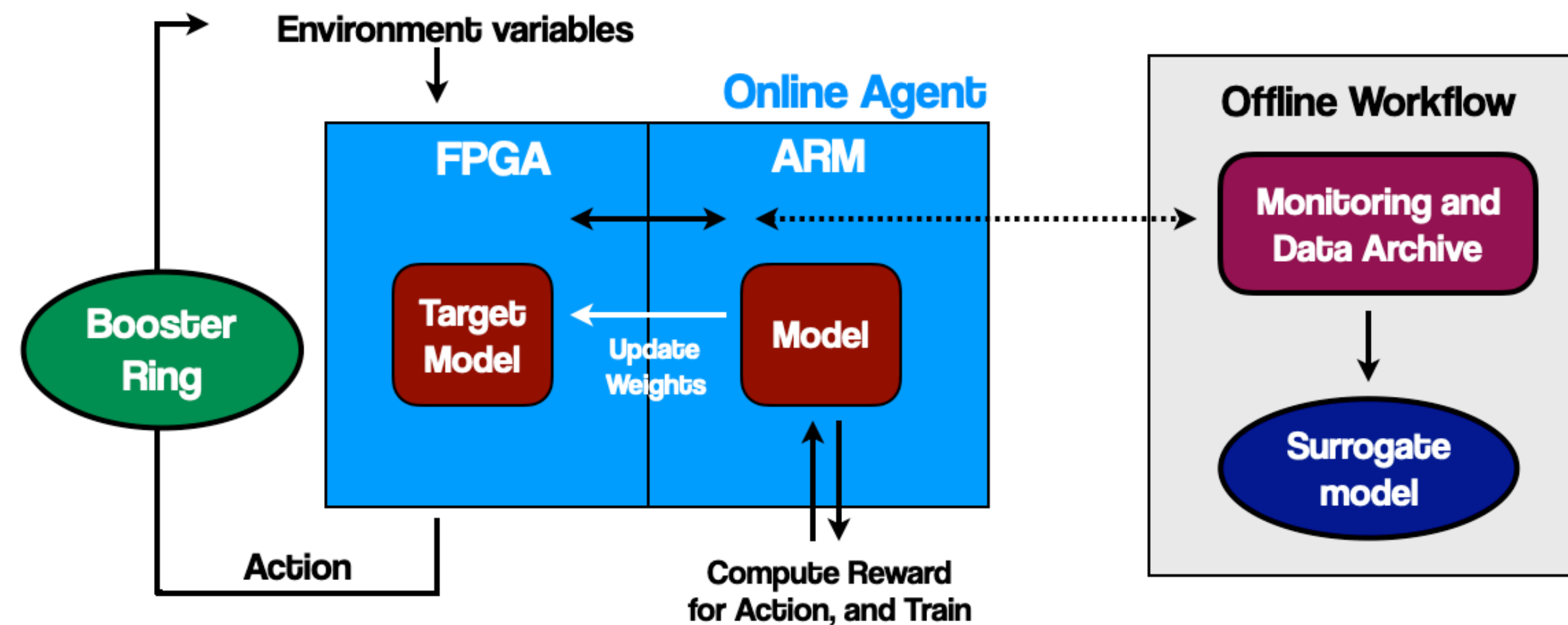
LSTM-based network using Deep Q-learning framework for surrogate model to mimic behavior of Booster

Challenges

Configuration like this with reconfigurable weights can be expensive, what about partially reconfigurable weights/activations?

e.g. tiny transfer learning, <https://arxiv.org/abs/2007.11622>

Data movement is expensive!
Can training data be transient?
Should we put the training hardware closer?



Is an optimized implementation able to generalize to all detector/accelerator conditions?

e.g. <https://arxiv.org/abs/2102.11289>

Outlook

- Real-time ML deployment still evolving — but it's very promising!
 - A lot of progress, quickly
- In rapidly moving space, considerations for system design that is both performant but flexible
- Examples given from sensor integration front-end, to FPGA filter stack, to coprocessors
- How to balance optimized performance and hardware implementations with generalizability and interpretability?